Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Dejan Jusufović

Grafi scene v igrah

diplomsko delo

na univerzitetnem študiju
računalništvo in informatika

Mentor: izr. prof. dr. Peter Peer

Ljubljana, 2016

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Grafi scene v igrah

Tematika naloge:

Predstavite grafe scen. Implementirajte tipične predstavnike grafov scen. Preizkusite jih v različnih tipih svetov, ki se pojavljajo v modernih igrah. Predstavite metrike za primerjanje uspešnosti grafov. Predstavite analizo uporabnosti grafov na reprezentativnih primerih.

Zahvaljujem se izr. prof. dr. Petru Peeru, za pomoč in potrpežljivost pri izdelavi naloge. Poleg tega bi se še zahvalil partnerki Poloni Kodermac za spodbudo.

Kazalo

Slike

Seznam uporabljenih kratic

kratica	angleško	slovensko
ABT	adaptive binary tree	adaptivno binarno drevo
PVS	potentially visible set	potencialno vidna množica
FPS	frames per second	število okvirjev na sekundo
	first person shooter	prvo osebna streljaška igra
BSP	binary space partitioning	binarno razdeljevanje prostora
RPG	role playing game	igra igranja vlog
PS3	Play station 3	konzola Play station 3
DLL	dynamic link library	dinamična knjižnica
MSPF	miliseconds per frame	milisekunde na okvir

Povzetek

Naslov: Grafi scene v igrah

Področje grafične tehnologije je bilo vedno omejeno s performančnostjo upodabljanja navideznih svetov. Toda performančnost ni odvisna samo od zmogljivosti strojne opreme, ampak tudi od načina specificiranja objektov, ki niso vidni. Najbolj preprosta rešitev tega problema vključuje ugotavljanje vidnosti za vsak objekt posebej, kar pri obsežnih navideznih svetovih predstavlja veliko časovno zahtevnost. Za premostitev navedenih problemov so bili razviti novi postopki, ki jih imenujemo grafi scene. Najbolj pomembna lastnost te skupine algoritmov je razdeljevanje navideznega prostora v zaključene celote z jasnim obsegom. Na takšen način se vprašanje vidnosti objektov poenostavi na vprašanje vidnosti področij, kar je lahko veliko bolj učinkovito. Cilj diplomskega dela je primerjava učinkovitosti grafov scen v različnih tipih svetov, ki se pojavljajo v modernih igrah. Pokazali smo, da vsak od uporabljenih grafov scen zelo poveča performančnost na vsaki obliki sveta, vendar obstajajo med grafi precejšne razlike. Ugotovili smo, da je najbolj pomemben faktor pri hitrosti grafov scen predvsem položaj in usmerjenost kamere ter struktura sveta.

Ključne besede: grafi scene, Octree, adaptivno binarno drevo, potencialno vidna množica, portali, 3D grafika, Ogre3D, igre, svet iger, razvoj iger.

Abstract

Title: Scene graphs in games

Graphics technology has always been limited with the rendering performance of virtual worlds. Performance is dependent not only on the hardware capabilities, but also on the specification of non-visible objects. The simplest solution requires detecting visibility for each object individually, which can be very time consuming, especially in large virtual worlds. To overcome these problems, new kind of algorithms were developed, which are called scene graphs. The most common trait of this group of algorithms is space subdivision into well defined regions. With this process the question of object visibility is transformed into region visibility, which can be a lot more efficient. The aim of the thesis is to compare various scene graphs in different worlds that occur in modern video games. We showed that each of the scene graphs used increased performance drastically in each world, but there is a lot of differences between them. We also observed that the most important factors influencing scene graph performance are camera position, direction and world composition.

Keywords: scene graphs, Octree, adaptive binary tree, potentialy visible set, portals, 3D graphics, Ogre3D, video game, video game world, game development.

Poglavje 1

Uvod

Kljub hitremu napredku grafične strojne opreme, bo potreba po programskemu odstranjevanju nevidnih objektov vedno obstajala. Ta izhaja predvsem iz želje igralcev in ustvarjalcev po vedno bolj obsežnih in podrobnih svetovih. Naloga obravnava grafe scen, s katerimi se lahko odstrani velik del nevidnih objektov ter razlike med njimi. V ta namen bodo implementirani najbolj pogosti predstavniki, na katerih bodo izmerjene različne metrike. Implementacija samih grafov scen je narejena znotraj odprto kodnega pogona Ogre3D [12].

1.1 Grafi scen

Navidezni svetovi, ki jih srečujemo v igrah in simulacijah, so zgrajeni iz velikega števila objektov, ki običajno ne nastopajo neodvisno od ostalih. Med njimi lahko obstaja vrsta relacij, med katerimi je najbolj osnovna prostorska relacija. Preprost primer takšne relacije lahko vidimo na primeru vozila. Ta je sestavljen iz množice ločenih objektov kot na primer šasija, kolesa in voznik. Premikanje takšnega vozila po prostoru je seveda možno z individualnim premikanjem vsakega od teh objektov, vendar to ne izkorišča implicitne hierarhije, ki je vsebovana v vozilu. Veliko lažje in intuitivno bi bilo premikanje samo enega objekta, na primer šasije, ki bi posodobila položaj vseh objektov, ki so od nje odvisni. Hierarhije objektov, ki se pojavljajo v navideznih svetovih, so lahko veliko bolj kompleksne od tega preprostega primera in za ta namen je bila razvita podatkovna struktura, imenovana graf scene. Graf scene je podatkovna struktura, ki na hierarhičen način povezuje

objekte, ki se nahajajo v prostoru. Drevo (kot primer grafa) je zgrajeno iz kopice vozlišč, pri čemer vsako vozlišče vsebuje informacijo, kako je razvrščeno v prostor glede na neposrednega predhodnika. Običajno je ta informacija podana v obliki afine matrike, ki opisuje translacijo, rotacijo in skalo. Takšna kombinacija matrik oziroma vozlišč od korena drevesa do trenutnega vozlišča določa pozicijo, rotacijo in skalo vozlišča. Običajno je graf scene direkten acikličen graf, kar pomeni, da cikli med vozlišči ne obstajajo ter da so le-ta povezana z usmerjeno povezavo. Takšna hierarhija omogoča pravilno pozicioniranje nekega vozlišča v primeru, da se spremeni prostorska informacija predhodnika. Na vozlišča, ki običajno vsebujejo samo matriko, se nato doda mrežne modele, luči in materiale, ki določajo vizualen izgled sveta. V igrah se pri najbolj preprosti izvedbi celoten graf posodobi po posamičnih vzorcih, in sicer naprej z algoritmom v globino. Posodabljanje poteka od korena drevesa do listov. Iteracija po drevesu lahko poleg posodabljanja prostorske informacije opravlja še druge funkcije. Hierarhija je lahko zgrajena na takšen način, da upošteva tudi upodabljanje. Na primer neko vozlišče, na katerega je povezan material, lahko pod sabo vsebuje samo mrežne modele, ki uporabljajo ta material. Najbolj pogosto pa je graf nadgrajen, tako da vsebuje tudi informacijo o obsegu objektov. Korensko vozlišče vsebuje celoten obseg sveta, ki predstavlja unijo obsegov naslednikov, ki si sledijo naprej vse do listov drevesa. Primarni razlog za računanje obsegov vozlišč je stožčasto izbiranje objektov, s katerim lahko odstranimo velike količine objektov, ki niso trenutno vidni. Vsak navidezni prostor je upodobljen preko kamere in njenega vidnega polja. Za pravilen prikaz sveta je zadostno samo prikazovanje objektov, ki so trenutno vidni. Vidnost pa se lahko ugotovi na preprosti način, in sicer s presekom obsega objektov in vidnega polja kamere [26].

V prejšnjemu odstavku je bilo govora tudi o drugi funkciji, ki jo lahko opravljajo grafi, in sicer o razdeljevanju prostora. Čeprav se v osnovi graf scene ukvarja bolj z odnosi med objekti, ki se nahajajo v svetu, je velikokrat njegova funkcija razširjena tudi na razdeljevanje prostora. Deljenje prostora prinaša ogromno koristi na številnih področjih, kot so na primer: odstranjevanje, detekcija trkov in razširjanje zvoka. Razdeljevanje prostora je v veliki večini primerov hierarhična operacija, kjer se en večji prostor razdeli na več podprostorov, ki predstavljajo naslednike [26]. Izjemo predstavlja sistem portalov. Celotno hierarhijo prostorov

lahko nato predstavimo v obliki drevesa. Velikokrat se hierarhična urejenost podprostorov združuje z osnovnim konceptom grafa scene do te mere, da ju obstoječa literatura več ne ločuje [28]. Sama izgradnja grafov scene s prostorsko delitvijo je relativno počasna operacija in se najpogosteje zgradi vnaprej.

1.2 Svetovi v igrah

Z vidika ugotavljanja vidnosti lahko svetove, ki se pojavljajo v igrah, razdelimo na tri skupine: odpri, pol-odprti in zaprti svet. Vsaka od treh omenjenih skupin je opisana v nadaljevanju.

1.2.1 Odprti svet

Takšen tip sveta pogosto srečujemo v igrah igranja vlog (angl. Role playing game – RPG), 3D simulacijah in vse pogosteje pri prvo osebnih streljaških igrah (angl. First Person Shooter – FPS). V večini primerov kamera oziroma igralec ni omejen pri izbiri poti skozi svet, razen v bližini robov sveta, kjer se gibanje omeji, predvsem zaradi praktičnih razlogov. Igralcu hočemo preprečiti, da odide izven igralne površine, saj lahko v tem primeru notranja logika igre odpove. Klasičen primer je detekcija trkov, s pomočjo katere se nastavlja položaj igralca, običajno po y osi, medtem ko potuje po svetu. Ta se mora prilagajati spremembam v višini površine, da dosežemo iluzijo hoje po svetu. Običajno detekcija trkov izven omejitev sveta ni definirana in povzroči, da entiteta, ki jo kontrolira igralec, prične padati in posledično tudi ponovni zagon igre. Drugi razlog za omejitev gibanja je, da nočemo opominjati igralca tekom igre, da je svet samo navidezen, saj je na robovih sveta zelo opazno, da izven omejitev sveta ni upodobitev. Oba problema se rešuje, tako s postavljanjem mrežnih modelov, ki omejujejo pogled igralca izven sveta in služijo tudi za omejevanje gibanja preko detekcije trkov, kakor tudi preko tako imenovanih nevidnih sten. Te so implementirane kot geometrija, ki se ne upodablja, ampak lahko še vedno opravlja funkcijo detekcije trkov. Največji tehnični izziv takšne oblike sveta je veliko in odprto področje, kjer je v najslabšem možnem primeru viden velik del sveta z malo medsebojnega prekrivanja objektov. Trenutno najbolj aktualne igre, ki uporabljajo takšen svet so Fallout 4 [6], Witcher3 [9], GTA V [16] in Skyrim [19] (slika 1.1).

Slika 1.1: Skyrim.

1.2.2 Pol-odprti svet

Igralec oziroma kamera sta v pol-odprtem svetu omejena na nek zaključen prostor z redkimi pogledi na celoten svet. Pogosto se takšen tip sveta pojavlja v 3D streljaških igrah in RPG igrah. Ker je v primerjavi z odprtim svetom igralec veliko bolj omejen, je na voljo bistveno več računalniških virov za doseganje boljše grafične predstavitve. Tipični predstavniki iger, ki uporabljajo samo takšen svet načeloma ne obstajajo, saj srečujemo takšne poglede na svet tudi v zaprtemu in odprtemu tipu sveta. Omenjeni tip sveta predstavlja vmesno točko med ostalima dvema in je bil dodan predvsem za testiranje razlik v performančnosti.

1.2.3 Zaprti svet

Zaprti svet je najbolj pogosta oblika sveta pri streljaških igrah. Zaprti svet ne pomeni nujno, da svet sestavljajo samo koridorji v obliki tunelov, ampak je to svet, kjer je gibanje igralca oziroma kamere močno omejeno. Velikokrat so igre narejene tako, da dajejo iluzijo odprtega sveta, čeprav je gibanje skozi svet linearno začrtano. Najbolj aktualne igre, ki uporabljajo takšen tip sveta so trenutnoo Battlefield1 [5], Overwatch [14] in Uncharted 4 [22] (slika 1.2).

Slika 1.2: Uncharted 4.

1.3 Uporaba v modernih igrah

Grafi scene so z razvojem doživeli veliko iteracij, ki so jih prilagajale trenutnemu stanju grafične tehnologije. Danes je največ poudarka na grafih scene, ki se lahko preprosto prilagodijo več procesorskim sistemom, vključno z efektivnim dostopom do pomnilnika. Ampak sama narava drevesnih struktur ni ravno pisana na kožo paralelnemu izvajanju [4]. Informacije o implementaciji grafov scene so skope in so večinoma posredovane preko člankov o odstranjevanju, kar pa ni povsem isto. V nadaljevanju so opisane rešitve, ki so jih uporabili pri nekaterih izmed najbolj znanih iger.

1.3.1 Uncharted: Drake's Fortune

Omenjena igra velja za eno izmed najbolj priljubljenih in znanih tretje osebnih streljaških iger podjetja Naughty Dog, ki je bila izdana ekskluzivno na konzoli PS3. Igra je razdeljena na več zaključenih svetov, kjer se dogajanje redko omejuje zgolj na notranjost. Vsak svet je predstavljen s PVS grafom, ki razdeli svet na več podprostorov. Vsebina oziroma objekti podprostorov pa so določeni z vrtenjem kamere na vzorčnih točkah in zajemanjem trenutnega okvirja. S procesiranjam

okvirjev so določeni vidni objekti. Za večjo natančnost tega postopka je bila dodana tudi možnost ročnega postavljanja vzorčnih točk. Rešitev je zaradi časovne zahtevnosti izdelave upoštevala prejšnjo verzijo grafa in s tem pohitrila postopek [8].

1.3.2 Battlefield 3

Battlefield 3 je zelo uspešna prvo osebna streljaška igra podjetja Dice, ki jo lahko srečamo na skoraj vseh platformah. Uporablja posebno obliko grafa scene, kjer je celoten svet razdeljen na bloke enake velikosti, ki skupaj tvorijo linearen seznam. Odstranjevanje je zaporedni postopek, ki najprej iterira preko blokov in nato preko objektov v blokih, za katere se ugotovi vidnost. Ker so bloki medsebojno neodvisni, je več nitna implementacija z malo vejitvami preprosta. Naknadno se za dodatne odstranitve pri rezultatih odstranjevanja uporabi še programska detekcija prekrivanja [4].

1.3.3 Witcher 3

Poleg igre Skyrim je Witcher 3 najbolj priljubljena fantazijska igra, ki se odvija v odprtem svetu. Pogon uporablja specializirano orodje Umbra za ugotavljanje vidnosti. Istovrstno orodje, ki deluje na osnovi sistema portalov, je bilo med drugim uporabljeno tudi v igrah Call of Duty: Ghosts, Destiny in World of Tanks. Posebnost orodja je v tem, da je postopek avtomatski in zahteva od uporabnika samo, da poda vsebino, ki se naj upodablja. Na podlagi vsebine, ki se naj upodablja, se zgradi zelo natančen sistem portalov, ki upošteva odprtine in možne prehode med področji [9, 21].

1.4 Struktura naloge

V 2. poglavju so predstavljeni načini ugotavljanja vidnosti, ki so bili uporabljeni v nalogi. Vsak graf scene je natančno opisan, vključno z njegovimi možnimi različicami. Predstavljene so tudi slabosti in prednosti grafov ter njihov vpliv na performančnost.

V 3. poglavju je opisano delovno okolje, s pomočjo katerega so bili izvršeni scenariji testiranja. Opisana je celotna pot, in sicer od izbire ustrezne vsebine, njene priprave ter uporabe. Podrobno so opisana uporabljena orodja ter testna aplikacija, s katero so bili izmerjeni rezultati.

V 4. poglavju so za vsako obliko sveta posebej določeni scenariji testiranja grafov scen. Razložene so motivacija pri definiciji scenarija in lastnosti vsake oblike sveta.

V 5. poglavju so za vsak graf scene na vseh uporabljenih svetovih predstavljeni rezultati odstranjevanja in upodabljanja.

V 6. poglavju sledi zaključek, kjer so predstavljene možne izboljšave.

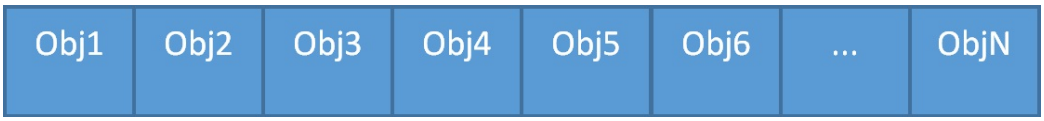
Poglavje 2

Grafi scen

V uvodnem poglavju je bila opisana splošna ideja grafov scen in motivacija za njihov pojav. V tem poglavju pa so navedeni podrobni opisi najbolj poznanih grafov scen in njihove posebnosti. Grafe scen delimo na tri večje skupine: sistem portalov, potencialno vidna množica (angl. Potentially Visible Set – PVS) in grafe binarnega razdeljevanja prostora [31], ki vsebuje več različic, tudi adaptivno binarno drevo (angl. Adaptive Binary Tree – ABT), Octree, ohlapni Octree in algoritem binarnega razdeljevanja prostora (angl. Binary Space Partitioning – BSP), po katerem je poimenovana celotna skupina algoritmov.

2.1 Linearen seznam

Za ugotavljanje vidnosti lahko uporabimo veliko postopkov, pri čemer je najbolj osnoven linearen seznam. Glavni lastnosti takšnega postopka sta odsotnost razdeljevanja prostora ter upoštevanje vseh objektov, ki se nahajajo v svetu. Edina uporabljena optimizacija je odstranjevanje na nivoju kamere, ki odstrani objekte izven vidnega polja kamere z namenom povečanja performančnosti upodabljanja. Linearen seznam objektov predstavlja v performančnosti spodnjo mejo, od katere se s tehnikami razdeljevanja prostora in ugotavljanja vidnosti poskušamo čim bolj odmakniti. Objekti so organizirani v navaden seznam, ki se ga obdela vsak okvir (slika 2.1). V določenem okvirju se zaporedoma preveri vidnost vsakega objekta v seznamu, in sicer s testiranjem vidnosti obsega objekta z vidnim poljem kamere. Največja odlika algoritma je predvsem preprostost implementacije.



Slika 2.1: Linearen seznam.

2.2 Octree drevo

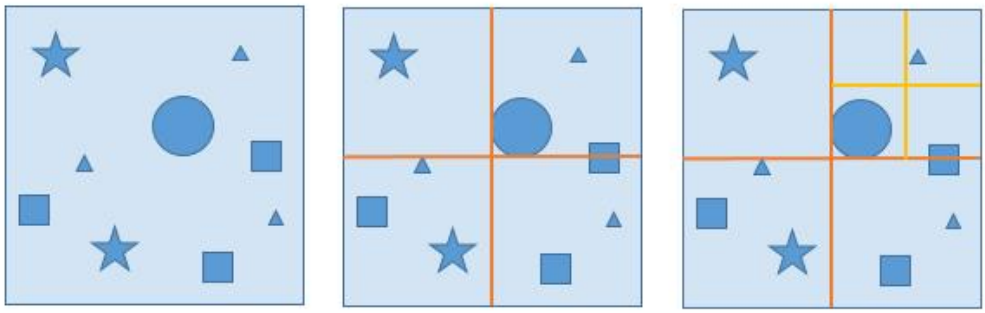
Octree algoritem je eden od možnih načinov razdeljevanja prostora za potrebe hitrega odstranjevanja velikega števila objektov. Osnovna ideja postopka temelji na enakomernem rekurzivnem razdeljevanju prostora, ki ga zajame svet, pri čemer se vsak objekt označi s podprostorom, ki mu pripada. Na podlagi te strukture in s primerjanjem mej podprostorov lahko na hiter način ugotovimo vidne objekte in odstranimo objekte izven vidnega kota kamere [26, 28].

Pri postopku najprej obkrožimo celoten svet v pravokotno kocko, ki tesno objema meje sveta. Nato pričnemo z drobljenjem kocke na manjše kocke z ravninami, ki se sekajo v centru prostora. To ustreza prepolovitvi začetne kocke na vsaki izmed osi x, y in z. Rezultat te operacije je 8 enako velikih kock, ki se imenujejo oktanti (slika 2.2). Pri vsaki delitvi se objekte označi z oktantom, ki mu objekti pripadajo. Ta postopek se ponavlja rekurzivno za vsak oktant posebej, dokler se ne doseže pogoj za izstop iz rekurzije. Pogoj za izstop iz rekurzije lahko predstavlja dosežena globina ali pa število poligonov znotraj kocke, ki je padlo pod določen minimum (slika 2.3).

Octree algoritem se lahko poenostavi v 2D obliko, ki se pogosto uporablja v 2D igrah z imenom Quadtree. Edina razlika je v tem, da se ena od osi preprosto zanemari, kar pomeni, da so rezultati deljenja 4 kvadranti in ne 8 oktantov.

Dobra stran Octree algoritma je regularnost in preprostost, saj so točke delitve vnaprej znane in jih ni potrebno izračunati. Vendar ima lahko preprostost tudi negativen učinek, saj algoritem ne upošteva dejanske geometrije v svetu, kot pri postopku ABT, in posledično, ni sposoben prilagajati točke delitve za vsako os posebej.

Ugotavljanje vidnosti pri Octree drevesu je tudi rekurziven postopek, ki se prične s korenskim oktantom. Za vsak oktant se najprej ugotovi njegova vidnost in v kolikor je vidnost pozitivna, se preveri vidnost vseh objektov znotraj oktanta, ki ji sledi rekurzivno preverjanje vseh podoktantov (slika 2.4). Algoritem je soroden



Slika 2.2: Quadtree kvadranti (če dodamo še tretjo os dobimo Octree oktante).

```
void IzgradiOctree(Obseg)
{
  if (globina > maxGlobina ||
      steviloPoligonov < minSteviloPoligonov)
  {
    return;
  }

  RazdeliObsegNaOktante();
  DodeliObjekteOktantom();

  for (int i = 0; i < 8; i++)
  {
    IzgradiOctree (ObsegOktanta(i);
  }
}
```

Slika 2.3: Pseudo koda izgradnje Octree drevesa.

postopku iskanja v globino.

Pri deljenju prostora na oktante se lahko pojavi problem, če določen objekt

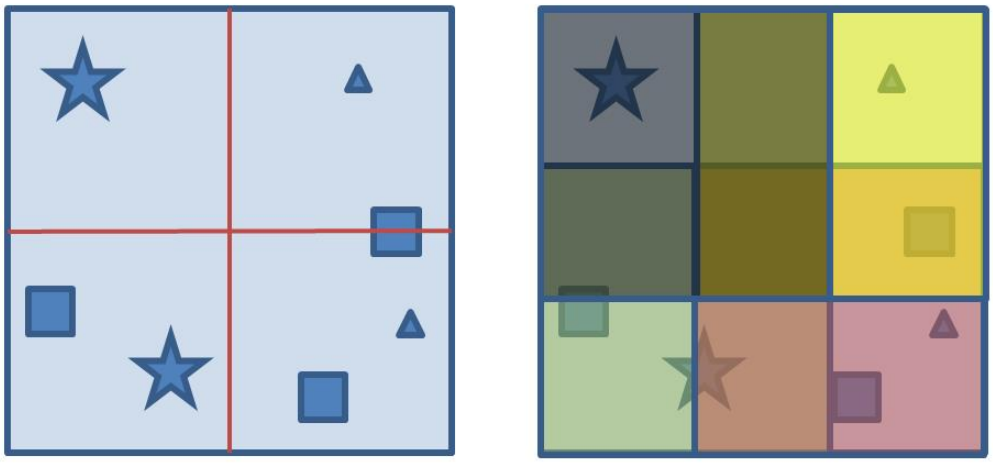
```
void VidnostOctree(Obseg)
{
    if (!Viden(Obseg))
    {
        return;
    }

    PreveriVidnostObjektov();
    for (int i=0; i<8; ++i )
    {
         VidnostOctree(ObsegOktanta(i))
    }
}
```

Slika 2.4: Pseudo koda za ugotavljanje vidnosti pri Octree drevesu.

pripada več oktantom hkrati. Ena izmed možnih rešitev je, da se objekt shrani v oba oktanta, in se pri upodabljanju uporabi zastavica, ki označuje, da je objekt že bil upodobljen. Slabost takega načina je dodatna kompleksnost in ne intuitivna struktura drevesa Octree. Druga rešitev je, da takšen objekt vstavimo v najmanjši oktant, ki zajema celoten objekt, kar pa privede posledično druge slabosti. Na primer, nek objekt bi lahko bil v centru scene, kar pomeni, da bi se vstavil v korenski oktant in bi posledično preverjanje vidnosti moralo v vsakem vzorcu vedno ugotavljati vidnost tega objekta. Omenjene probleme lahko preprosto rešimo, tako da vse objekte, ki so na presečišču ravnin, preprosto prerežemo na pol in dodelimo polovici ustreznim oktantom. Pri takšni rešitvi se lahko pojavi znatno povečanje objektov v svetu in se posledično zveča zahteva po odstranjevanju in upodabljanju [26].

Za premostitev zgoraj naštetih problemov lahko posežemo po ohlapnemu Octree algoritmu, ki je med drugim uporabljen tudi v tej nalogi. Posebnost omenjenega algoritma je v tem, da meje oktantov niso fiksne, ampak so povečane, običajno za faktor 1.5, objekt pa se vedno vstavi le v en oktant. Posledično se

bistveno manj objektov pojavlja na presečišču ravnin, zaradi česar jih lahko potisnemo globlje v hierarhijo. Na sliki 2.5 je primerjava oktantov običajnega Octree drevesa in ohlapne različice s skalirnim faktorjem 1.5 [26].



Slika 2.5: Razlika med Quadtree in ohlapnimi Quadtree kvadranti.

2.3 Adaptivno binarno drevo

Tudi ta postopek spada med postopke binarnega razdeljevanja prostora, vendar se od ostalih sorodnih postopkov ločuje zaradi bistvene lastnosti. ABT razdeljuje prostor na podlagi geometrije, nad katero operira. Celoten prostor se, podobno kot pri postopku Octree, razdeli s pomočjo ravnin, ki so pravokotne na glavne osi, vendar položaj ravnin vzdolž glavnih osi ni vnaprej znan, ampak se izračuna z upoštevanjem določenih parametrov. Primer omenjenih parametrov je število presekanih poligonov, čemur se običajno izogibamo. Zaradi prilagodljive narave postopka se ABT ne glede na obliko sveta odreže dobro. Rezultat deljenja prostora z ravnino sta natanko dva podprostora.

Preden lahko uporabimo ABT, je potrebno za celoten svet določiti meje, ki tesno objemajo svet. Nato se izbere ena izmed osi, na katero bo pravokotna ravnina, ki bo prostor razdelila na pol. Rezultat preseka sta dva ločena, zaključena prostora vzdolž izbrane osi, ki se lahko uporabita v naslednjih iteracijah algoritma.

Postopek se rekurzivno ponavlja, dokler se ne doseže določena globina, oziroma dokler število poligonov postane premajhno.

Najbolj pomemben del algoritma je izbira glavne osi za presek ter lokacija na osi, skozi katero bo potekala ravnina. S pretvorbo v minimizacijski problem, kjer se išče največji minimum linearne enačbe specifičnih kriterijev, postopek ponuja odgovor na obe vprašanji. Rešitev se lahko zapiše v obliki enačbe na sliki 2.6, kjer rezultat predstavlja ustreznost neke ravnine pri razdeljevanju [27].

$f_1 =$ največja velikost prepolovljenih osi

$f_2 =$ volumen nastalih prostorov

$f_3 =$ število poligonov v nastalih prostorih

$f_4 =$ število presekanih poligonov

$$\text{f(položaj)} = w_1 * f_1(\text{položaj}) + w_2 * f_2(\text{položaj}) + w_3 * f_3(\text{položaj}) + w_4 * f_4(\text{položaj})$$

$w_1 =$ utež parametraf_1

$w_2 =$ utež parametraf_2

$w_3 =$ utež parametraf_3

$w_4 =$ utež parametraf_4

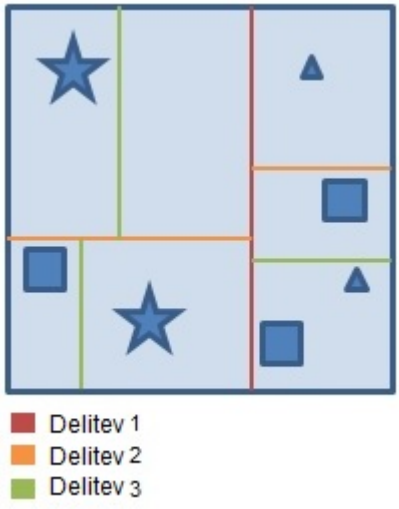
Slika 2.6: Parametri izgradnje ABT drevesa.

Glede na rešitev, ki jo išče, mora uporabnik sam določiti uteži za vsak posamezen kriterij. Na primer, če ne želimo ustvariti novih poligonov, nastavimo četrto utež blizu 1, če želimo enakomerno razporejenost poligonov po novo nastalih prostorih, nastavimo tretjo utež blizu 1, in tako naprej.

Prvi parameter vrača maksimum velikosti prepolovljenih glavnih osi pri dani točki, kar pomeni, da iskanje minimuma favorizira ravnine, ki presekajo največjo os trenutnega prostora. Volumen nastalih prostorov predstavlja razliko v volumnu novo nastalih prostorov in favorizira ravnine, ki drobijo originalni prostor na približno enako velike volumne. Parameter število poligonov vrne razliko števila

poligonov v nastalih prostorih z namenom enakomerne porazdelitve poligonov, medtem ko število presekanih poligonov predstavlja število poligonov, ki nastanejo po preseku z ravnino.

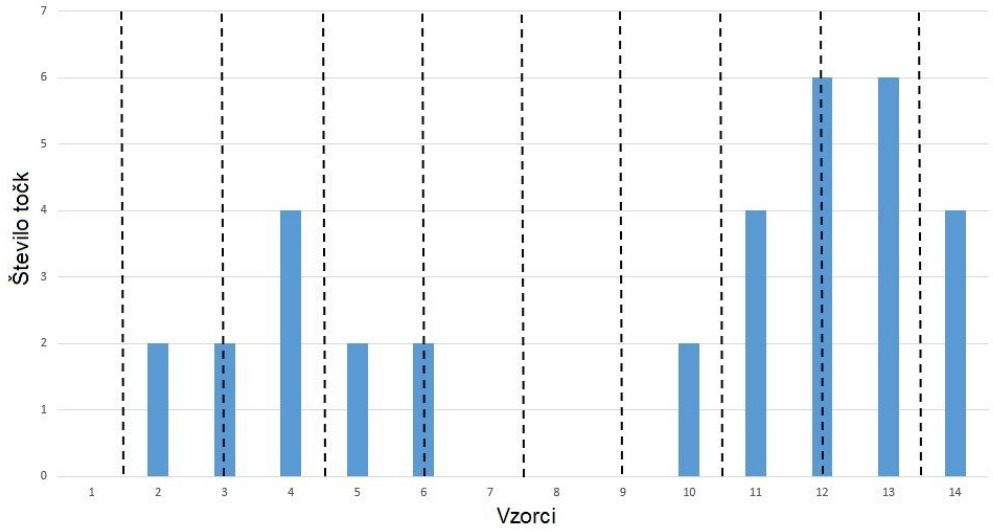
Z zgoraj opisanim postopkom lahko izračunamo ustreznost določene ravnine, vendar ostaja še vedno problem izbire točke, skozi katero poteka ta ravnina. Točka se lahko nahaja na poljubni osi. Za izbiro točk je bilo razvitih več postopkov, od katerih je najbolj preprost postopek linearnega vzorčenja. Na sliki 2.7 je prikazana ena izmed možnih oblik ABT delitve. V primerjavi z Octree drevesom je delitev zelo ne regularna, saj upošteva dejansko lego geometrije pri izbiri ravnin, ki razpolavljajo prostor.



Slika 2.7: Deljenje prostora po principu ABT.

Pri postopku linearnega vzorčenja vsako izmed osi enakomerno vzorčimo in po dani enačbi na pridobljenih točkah v prostoru izračunamo rezultat. Ravnina, ki ima najmanjšo vrednost, je izbrana za deljenje prostora. Postopek je zelo preprost, vendar ima slabost, saj lahko zgreši področja v svetu, kjer je geometrija koncentrirana. Na sliki 2.8 je prikazano, kako izgleda omenjen postopek na eni izmed osi. Višina stolpca predstavlja število točk oziroma verteksov na določenem položaju osi x, medtem ko navpične črtkane črte predstavljajo možne lokacije ali vzorce za izbiro ravnine.

Postopek, ki je bil uporabljen v tem delu, se imenuje statistično uteženo

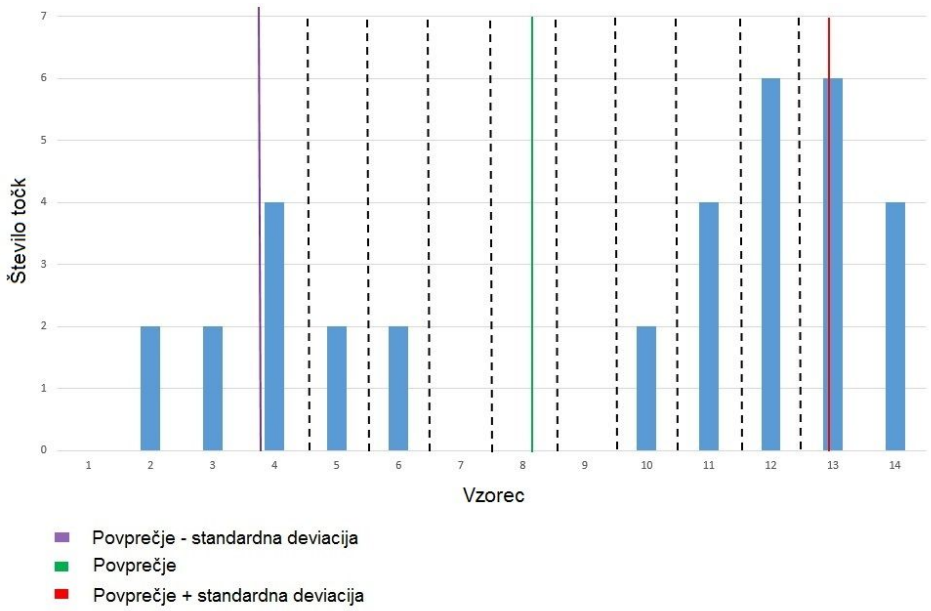


Slika 2.8: ABT linearno vzorčenje.

vzorčenje, ki za ceno večje kompleksnosti premošča probleme linearnega vzorčenja. Algoritem za vsako izmed osi izračuna povprečno vrednost in standardno deviacijo ter vzorči točke na obeh straneh povprečja z razmakom največ ene standardne deviacije. Primer omenjenega postopka prikazuje slika 2.9, kjer je vzorčenje omejeno na okolico povprečne vrednosti. Za namen primerjave obeh verzij vzorčenja, oba načina uporabljata isto število vzorcev. Pri statističnem vzorčenju je iskanje ravnine veliko bolj osredotočeno in ponuja med drugim tudi možnost zmanjšanja potrebnih vzorcev, s čimer se postopek izgradnje ABT drevesa lahko pospeši.

2.4 Potencialno vidna množica

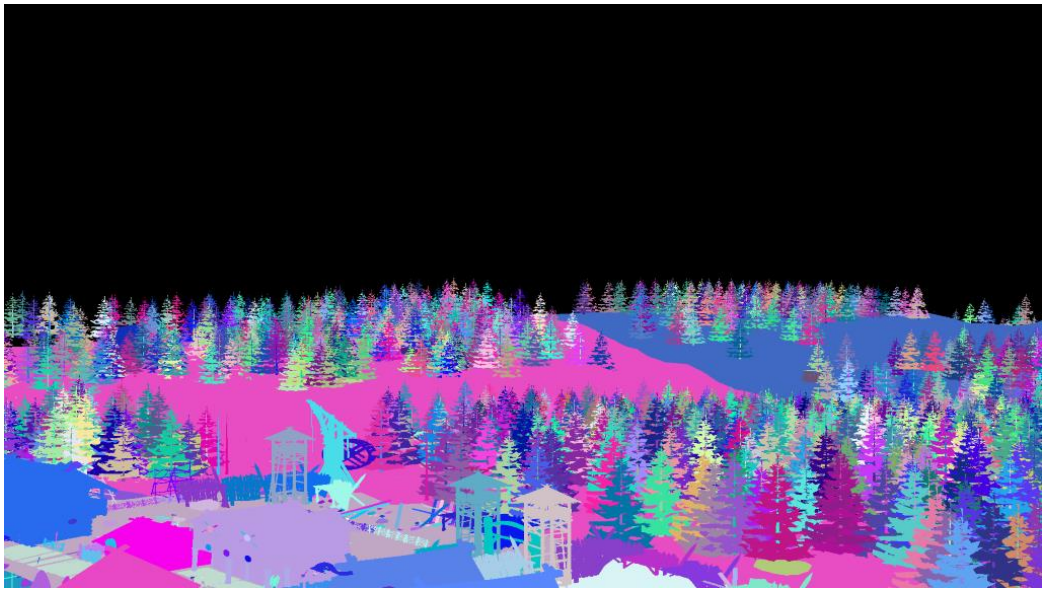
Algoritem PVS za razliko od predhodno omenjenih algoritmov upošteva prekrivanje objektov. Ideja postopka se zanaša na lastnost, da je večina sveta zakritega z ostalimi objekti in da je nepotrebno ugotavljanje vidnosti oziroma upodabljanje objektov v ozadju [10]. Postopek razdeli celoten svet, ne glede na njegovo zgradbo, na enako velike celice. Znotraj celic se nato izberejo položaji, kamor se postavi kamera. S postavljanjem in vrtenjem kamere na izbranih točkah, si algoritem zapomni vidne objekte in jih pripiše trenutni celici. Postopek se nato ponovi za vse celice. Vidnost znotraj celice je moč ugotoviti na več načinov. Ena izmed



Slika 2.9: ABT statistično uteženo vzorčenje.

možnosti je, da kamera projicira žarke v obliki stožca, preko katerih se zaznajo možna presečišča z objekti. Bolj enostavna in natančna metoda, ki je tudi uporabljena v tem delu, pa obarva vsak objekt z drugo barvo. Celotno sliko, ki je bila zajeta s trenutnim položajem kamere, se nato obdela in se preveri barvo vsake slikovne točke v sliki. Na podlagi barve se iz dane celice določi viden objekt (slika 2.10).

Glede na pripravo množic lahko postopke PVS v grobem delimo na dve skupini in sicer realno-časovno ter vnaprejšnjo. V prvem primeru je običajno zaradi časovne zahtevnosti algoritma število točk znotraj celice omejeno, medtem ko drugi te omejitve nima. Prednost vnaprejšnjih različic je predvsem v tem, da se velik del obdelave preseli izven izvajanja aplikacije, kar prinaša višjo performančnost. Takšna oblika je predvsem primerna za statične svetove, kjer je zelo malo premikajočih objektov, saj le-ti niso bili zajeti v postopek grajenja. Realno-časovno obliko pa se uporablja v bolj dinamičnih svetovih, četudi še vedno ostaja omejena s časovno zahtevnostjo algoritma. Ker se v nalogi ne pojavljajo dinamični objekti, je uporabljena vnaprejšnja različica. V nalogi je svet prav tako razdeljen na enako število celic vzdolž vsake osi.



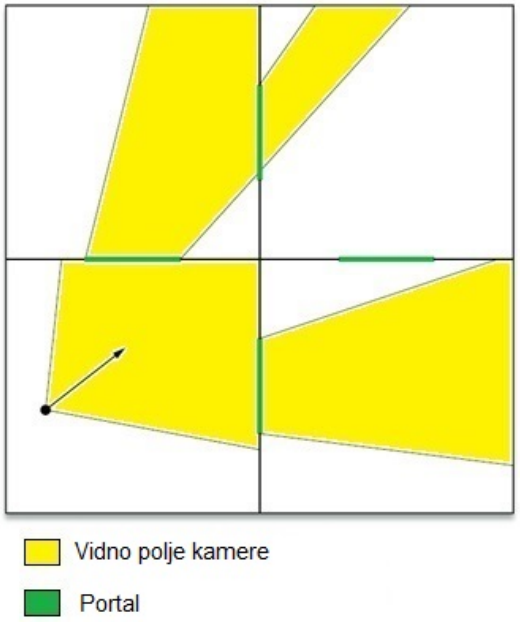
Slika 2.10: Obarvanje vsakega objekta z unikatno barvo pri gradnji PVS grafa.

2.5 Portali

Portal predstavlja kot algoritem za ugotavljanje vidnosti preprost in učinkovit način za določanje vidnosti sveta preko vidnega polja kamere. Osnovna ideja temelji na razdelitvi sveta na ločena področja oziroma cone, ki so povezana preko tako imenovanih portalov (slika 2.11). Portal je konveksni poligon, preko katerega lahko vidimo neko področje. V realnem svetu bi se portal recimo poistovetil z oknom ali vrati, preko katerih je vidna zunanjost. Pri ugotavljanju vidnosti se pokaže velika prednost portalov, saj pri upodabljanju upoštevamo samo področja, katerih portali so vidni [30].

Običajno se svet ročno razdeli na zaključene celote oziroma celice ali cone. Urejevalniki sveta vsebujejo vgrajena orodja za postavljanje portalov in povezovanje con. Vsakemu objektu, ki se nahaja znotraj cone, se doda informacija o pripadajoči coni. Na takšen način ima uporabnik popolno kontrolo in lahko prilagaja lokacijo portalov glede na posebnosti igre oziroma simulacije. Slabost takšnega načina je predvsem v zamudnosti, saj je potrebno cone oziroma portale popraviti pri vsaki spremembi sveta.

Algoritem za ugotavljanje vidnosti je zelo preprost. Najprej se ugotovi cona v kateri se nahaja kamera, ki se avtomatsko označi kot vidna, vključno z vsemi objekti, ki se nahajajo v njej. Nato se preveri, kateri portali so vidni preko vidnega polja kamere. Na podlagi vsakega vidnega portala in njegove oblike se vidno polje kamere omeji, tako da vidno polje potuje samo skozi ta portal. Z novim vidnim poljem se preveri vidnost objektov v coni, skozi katero vodi portal. Ta postopek ponovimo za vsakega izmed začetnih portalov. Postopek se nato nadaljuje z vidnimi conami iz prejšnjega koraka, upoštevajoč omejeno vidno polje kamere. Algoritem je rekurzivne narave in se izvaja, dokler so še vidni portali. Na sliki 2.11 je prikazan postopek omejevanja vidnega polja kamere, kjer črna pika in vektor predstavlja položaj kamere in njeno usmeritev. Rumena barva predstavlja vidno polje kamere, medtem ko zelena barva predstavlja portale.



Slika 2.11: Primer omejevanje vidnega polja kamere prek portalov.

Poglavje 3

Uporabljena orodja in testna aplikacija

3.1 Ogre3D

Začetki pogona Ogre3D segajo v leto 2001, in sicer sovpadajo z idejo ustanovitelja projekta Steve Streeting-a o drugačni obliki grafičnega pogona, ki bi premoščal pomanjkljivosti tedaj obstoječih pogonov. Glavna področja, na katere se je ustanovitelj osredotočil, so bila: obilo dokumentacije za vsak aspekt pogona, objektno usmerjena arhitektura, pogosta uporaba načrtovalskih vzorcev, graf scene, ki je popolnoma ločen od vsebine scene, možnost dodajanja nove funkcionalnosti preko vtičnikov ter poudarek na kvaliteti in ne kvantiteti zmožnosti pogona [29]. Prva različica s kodnim imenom Azathoth je izšla februarja 2005 in je do danes doživela že veliko iteracij. Prvotno je bil pogon objavljen z dvema licencama, bolj restriktivno LPGL in Ogre Unrestricted License, ki je bila namenja predvsem za svet konzol, kjer so restriktivne licence zelo nezaželene. Leta 2012 se je licenciranje spremenilo na MIT licenco, ki je zelo preprosta in ne omejujoča. Zadnja stabilna verzija pogona 1.9 je izšla leta 2013 in predstavlja tudi zadnjo verzijo Ogre3D, ki temelji na osnovno zastavljeni arhitekturi [12].

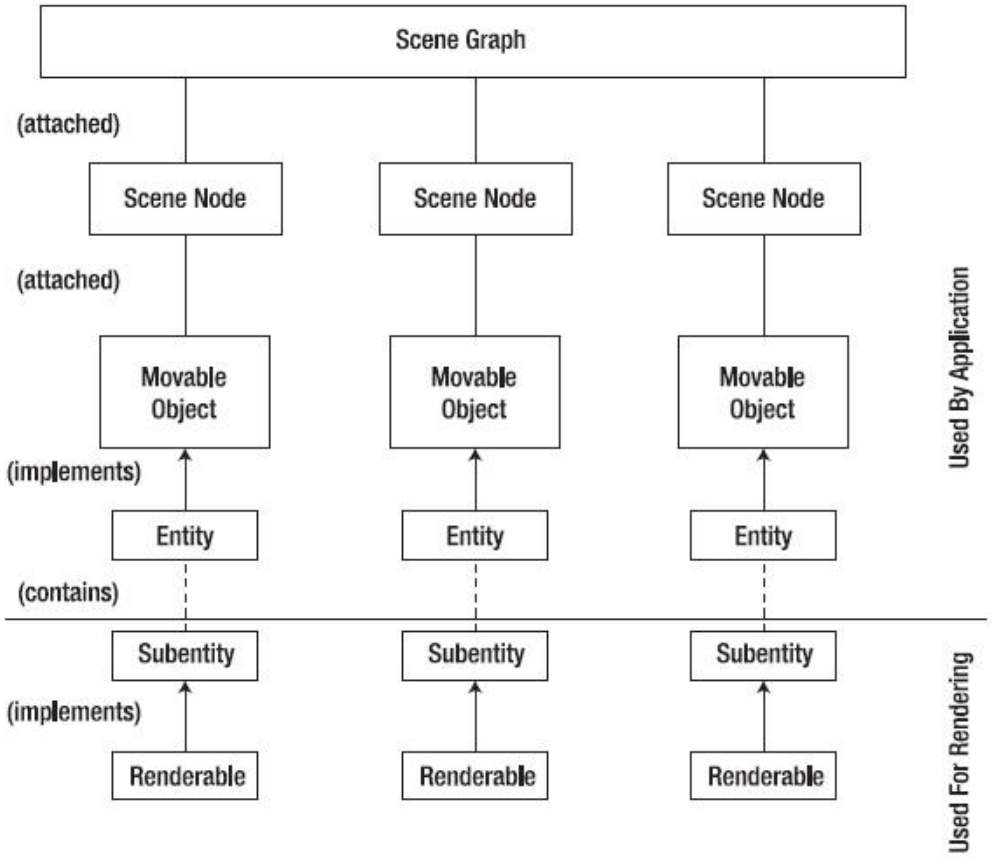
V zadnjem času poteka precejšna prenova pogona, saj se želi uskladiti arhitekturo z razvojem računalniške tehnologije v zadnjem desetletju [11]. Fokus prenove je programska arhitektura, ki je pisana na kožo več jedrnim procesorjem in upo-

rablja tehnike s področja podatkovno usmerjenega programiranja [3]. Omenjene tehnike so namenjene premostitvi vrzeli med hitrostjo procesorjev na eni strani ter pomnilnikom na drugi strani, ki se kljub razvoju tehnologije povečuje, ampak še vedno predstavlja t.i. zoženo grlo pri zmogljivosti pogona v njegovi trenutni obliki.

Za namene tega dela bo opisana arhitektura pogona (slika 3.1), ki omogoča dodajanje poljubnih grafov scen. Takšna funkcionalnost zahteva ločitev implementacije grafa scene od pogona in ločitev samega grafa scene od vsebine, ki se lahko pojavlja v svetu. Pod vsebino imamo v mislih predvsem objekte sledečih tipov: mrežni modeli, luči, delci, itd. Vsak nov graf scene mora ustrezati virtualnemu abstraktnemu vmesniku SceneManager, ki ga definira pogon, in preko katerega Ogre3D posreduje klice za posodabljanje ter upodabljanje grafa. Običajno je vsaka nova implementacija vmesnika narejena v obliki dinamične knjižnice (angl. Dynamic Link Library – DLL). Neodvisnost vsebine scene od same implementacije grafa je dosežena preko objektov tipa SceneNode, ki v osnovi definirajo transformacijo v prostoru in na katere se poveže objekte s vsebino. To pomeni, da ni univerzalnega razreda, iz katerega bi bili razširjena vsebina, ki bi neposredno povezovala vsebino z grafom scene. Namesto tega mora vsaka vsebina ustrezati abstraktnemu virtualnemu vmesniku MovableObject, preko katerega graf scene komunicira z dejansko vsebino oziroma objektom. Oboje skupaj pomeni, da lahko poljubno spreminjamo implementacijo algoritma grafa scene, brez da bi vplivali na implementacijo vsebine in z možnostjo poljubnega spreminjanja določene implementacije vsebine, brez da bi to vplivalo na graf scene. Primer takšne implementacije je razred Entity, ki predstavlja nekakšen mrežni model [29]. Kljub mnogim prednostim tekom svojega obstoja Ogre3D ni doživel večje priljubljenosti. Čeprav je zaradi svoje odprte narave priljubljen vir za učenje algoritmov in arhitekture, obstaja malo komercialnih produktov, ki temeljijo na njemu. Najbolj znani igri, ki sta bili osnovani na Ogre3D, sta Torchlight 1 in 2 [17, 18].

3.2 Maya

Autodesk Maya [1] je eden izmed najbolj razširjenih programov za 3D oblikovanje in izdelavo animacij v filmski industriji in vse pogosteje tudi v industriji iger.



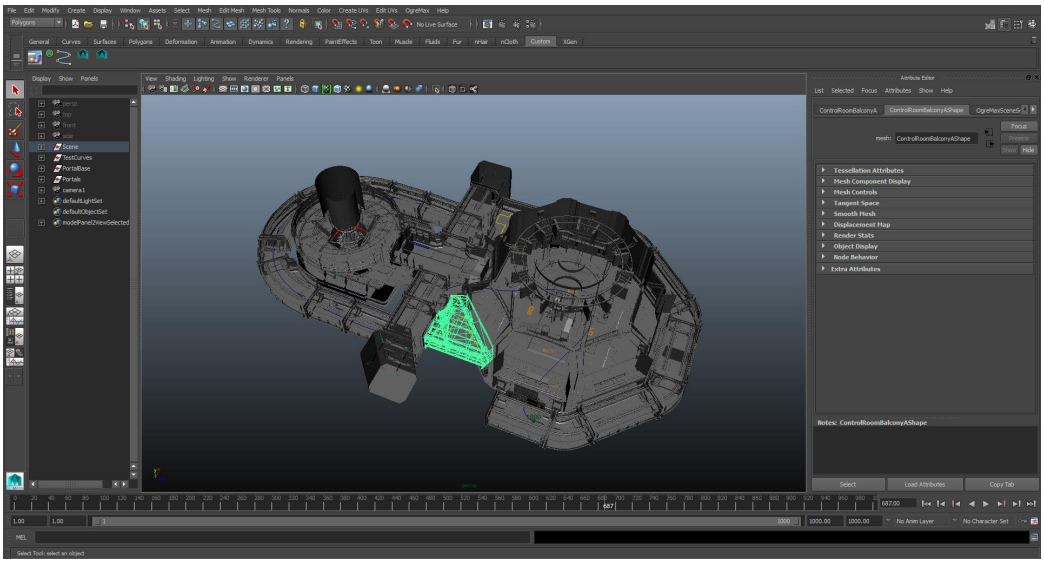
Slika 3.1: Ogre3D arhitektura.

Daljni začetki segajo v devetdeseta leta, ko sta se pojavila programa The Advanced Visualizer podjetja Wavefront in PowerAnimator podjetja Alias, s pomočjo katerih so bili narejeni posebni efekti v filmih kot so na primer: Jurrasic Park, Terminator 2 ter Independence Day. Na osnovi obstoječe kode programa PowerAnimator je podjetje Alias za namen produkcije filma Dinosaur skupaj z Disney-om razvilo prvo verzijo Maye, ki je izšla leta 1998. Mayo je odlikoval predvsem bogat in popolnoma prilagodljiv uporabniški vmesnik in možnost pisanja skript v skriptnem jeziku MEL. Leta 2005 je podjetje Autodesk prevzelo podjetje Alias, s čimer je Maya postala del večjega paketa orodij, ki jih ponuja Autodesk, in sicer orodij kot so 3D Studio Max, Houdini [2].

V zadnjem desetletju je izjemen napredek v industriji iger omogočil, da je

postala Maya eno izmed poglavitnih orodij, ki se jih poslužujejo tudi razvijalci računalniških iger. Z Mayo so bile narejene nekatere izmed najbolj uspešnih iger, kot na primer: God of War 3, Uncharted 2 ter Ratchet and Clank. Čeprav se Maya še vedno najpogosteje uporablja za izdelavo animacij in 3D oblikovanje, je programski paket dovolj bogat, da lahko v nekaterih primerih služi tudi kot urejevalec sveta. Primer slednjega je slovensko podjetje Zootfly in njihov pogon Zen.

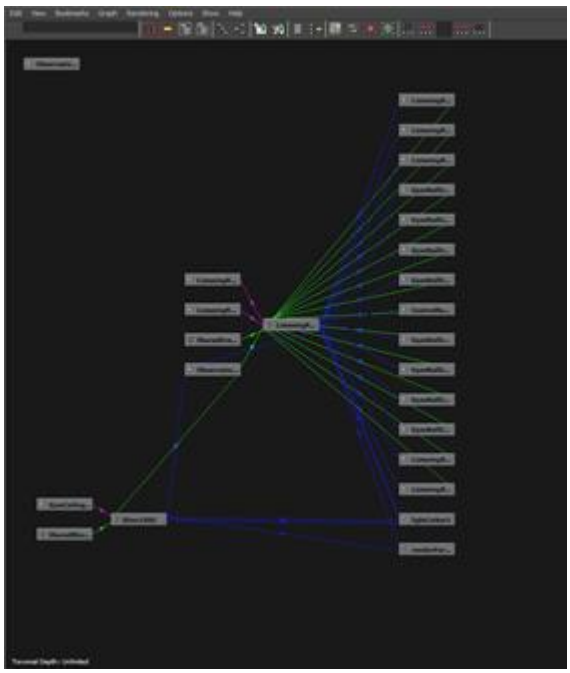
Največje prednosti programa Maya ležijo predvsem v bogati paleti orodij, ki zajemajo širok spekter področij, kot so na primer: 3D oblikovanje, izdelava animacij, fluidnih efektov, fizikalno realističnih animacij togih in mehkih teles, efektov delcev in tako naprej. Poleg tega je treba omeniti tudi ogromno količino dokumentacije, ki je prosto dostopna na uradni strani in preko forumov razvijalcev. Med slabosti programskega paketa pa spadajo zelo strma krivulja učenja in visoka cena, čeprav slednja ne predstavlja več bistvene prepreke, odkar je na voljo okrnjena različica za študijske namene.



Slika 3.2: Prikaz zaprtega sveta v Mayi, ki je tudi uporabljen v nalogi.

Delo znotraj Maye poteka tako, da uporabniki ustvarijo sceno, ki predstavlja virtualno delovno okolje, ki se običajno zapolni z mrežnimi modeli in animiranimi karakterji (slika 3.2). V ozadju je scena predstavljena z direktnim acikličnim grafom vozlišč, ki vsebujejo različne atribute, medtem ko povezave atributov vozlišč

določajo vse od hierarhične ureditve scene do grafične predstavitve objektov (slika 3.3).



Slika 3.3: Vozlišča v Mayi.

Maya je eno izmed orodij, za katero obstaja podpora uvoza v Ogre3D. Za ta namen je bilo razvitih več konkurenčnih programov, pri čemer je potrebno posebej omeniti program OgreMax [13], ki je velja za najbolj stabilnega z obilo možnosti izvoza vsebine. Program izpiše celotno Maya sceno v .scene datoteko, medtem ko v Ogre3D izvozi mrežne modele in materiale v ustreznem formatu datotek, in sicer oblike .mesh in .material. V .scene datoteki je zapisana hierarhija objektov, povezave do mrežnih modelov, materialov in tako naprej.

Interno je program ustvarjen kot vtičnik za Mayo, kar pomeni, da se s svojimi meniji in atributi lahko obnaša kot sestavni del Maye. Vsak ustvarjen objekt znotraj Maye pridobi avtomatsko zbirko nastavljivih OgreMax atributov, ki se preslikajo v atribute objektov znotraj Ogre3D pogona (slika 3.4). Vtičnik praktično spremeni Mayo v urejevalnik sveta za pogon, saj že Maya sama po sebi vsebuje kopico orodij za urejanje vsebine. Kljub temu ta orodja niso na istem nivoju, kot pri pogonu Unity, saj je bil program Maya razvit za druge namene. Omenjeno

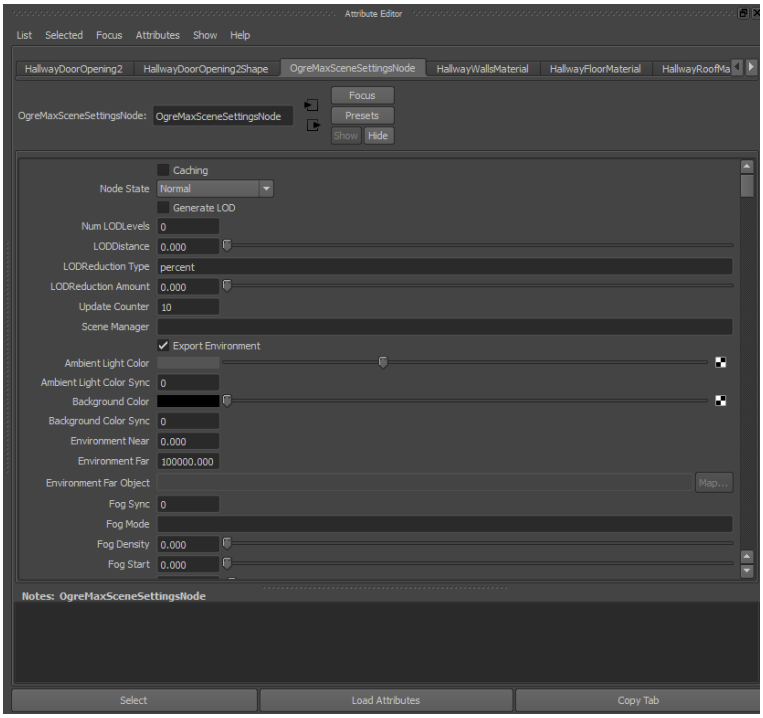
pomanjkljivost je moč premostiti preko skriptnega jezika MEL (slika 3.5), ki izpostavlja večino funkcionalnosti Maye. V ta namen so bile za potrebe olajšanja dela pri izvozu vsebine razvite sledeče MEL skripte:

- skripta za preimenovanje objektov, saj pogon Ogre3D za razliko od Unity3D, ne dovoljuje podvojenih imen
- skripta za odstranjevanje skaliranja iz transformacijskih matrik poligonov, ki predstavljajo portale, zaradi načina delovanja vtičnika portalov v Ogre3D
- različne skripte za masovno nastavljanje atributov OgreMax, kot na primer: izvoz normal, tangent na mrežnih modelih
- skripta za spremembo vrstnega reda evaluacije rotacije objektov
- skripta za izvoz krivulj
- skripte za pomoč pri pretvorbi Unity3D materialov
- skripta za poenostavitev izvoza celotne scene v Mayi z ugotavljanjem konsistentnosti vsebine

3.3 Unity3D

Pogon Unity3D [23] se uporablja samo za pridobitev svetov, saj predstavlja edini način za prenos zastonjske vsebine preko Unity Asset Store strani, ki je neposredno povezana s pogonom Unity3d (slika 3.6). Uvoz v Unity3D je zelo preprost in zahteva od uporabnika zgolj to, da ustvari svoj račun, nakar lahko uporabnik prosto izbere željeno vsebino na Unity Asset Store strani in prične s prenosom v pogon.

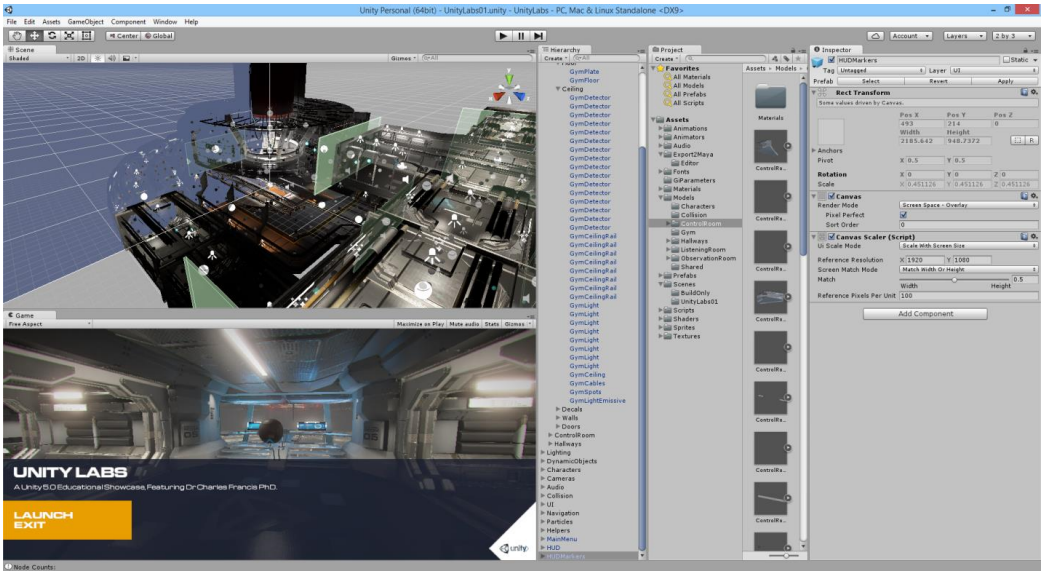
Prenos vsebine neposredno v Ogre3D trenutno ni podprt, obstajajo pa orodja za uvoz iz programov Autodesk Maya ali 3D Studio Max. Problem lahko rešimo s plačljivim vtičnikom za Unity3D z imenom Export2Maya [25], ki pretvori opis scene v pogonu Unity v format namenjen za program Maya. Rezultat vtičnika je Maya datoteka, v kateri so zapisani položaji vseh objektov, vključno z lučmi,



Slika 3.4: OgreMax atributi znotraj Maye.

```
global proc ExportTestCurves()
{
    select -hi "TestCurves";
    select `ls -type "nurbsCurve"`;
    string $curves[] = `pickWalk -d up`;

    int $i=0;
    for ($i=0;$i<size($curves);$i++)
    {
        ExportCurveEP($curves[$i]);
    }
}
```

Slika 3.5: Primer MEL skripte.



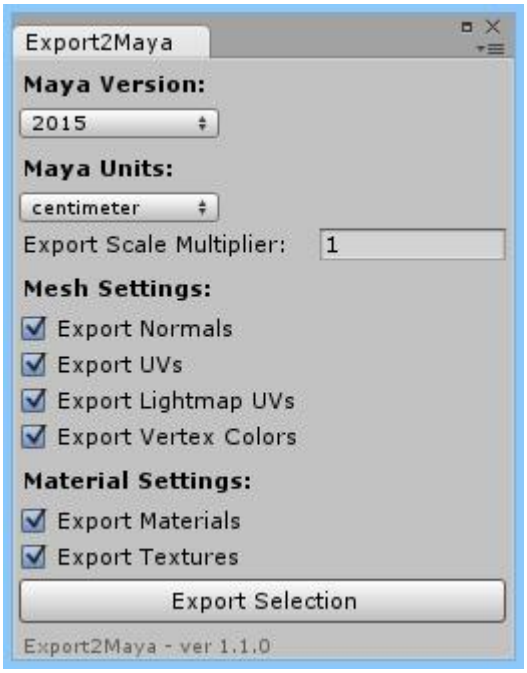
Slika 3.6: Unity3D okolje.

mrežnimi modeli in senčilniki. Poleg Maya datoteke so prav tako izvožene in shranjene uporabljene teksture. Izvoz zahteva samo, da v Unity3D označimo objekte, ki jih želimo izvoziti, nakar poženemo vtičnik Export2Maya (slika 3.7). Velja omeniti, da je proces izvoza zelo dolgotrajen proces in lahko v primeru velikih scen traja tudi uro ali več. Problemi vtičnika se pojavijo pri Unity3D materialih, ki jih ni moč neposredno pretvoriti v May-in ekvivalent in pri materialih, ki imajo več kot eno teksturo. Pri slednjem teksture ne bodo izvožene in jih je potrebno ročno poiskati v Unity projektni mapi, medtem ko se bolj zapleteni Unity3D materiali predstavijo v Mayi s približki, kot so na primer materiali lambert, blinn ali phong.

Vtičniku se lahko izognemo, vendar to v Mayi zahteva precej več časa za vzpostavitev identične scene. Unity3D hrani vso vsebino v sceni, kot na primer: mrežne modele, teksture in zvoke v mapi Assets pod trenutnim Unity projektom. Te bi lahko načeloma uvozili, enega po enega v Mayo, kar pa bi bilo precej zamudno, saj bi zahtevalo ročno postavljanje vsakega objekta v prostoru.

3.4 Aplikacija

Testiranje grafov scen ni mogoče brez realističnih svetov, vendar je ustvarjanje le teh zelo dolgotrajen postopek, ki bi presegal obseg tega dela. Namesto tega lahko



Slika 3.7: Export2Maya vmesnik.

uporabimo že ustvarjene svetove, ki so namenjeni za uporabo v drugih pogonih. Unity Asset Store tako ponuja več zastonjskih svetov, pod pogojem da je uporaba slednjih namenjena za nekomercialne in izobraževalne namene [24]. Ti predstavljajo merilo za zmožnosti in zmogljivosti Unity pogona in so idealni kandidati za testiranje grafov scen. Preko izvoznih programov Export2Maya in OgreMax lahko uvozimo takšne svetove v pogon Ogre3D, izgradimo grafe scene ter opravimo teste.

Aplikacija je namenjena testiranju grafov scene na različnih svetovih. Največji problem pri izdelavi aplikacije so bile različne zahteve, ki jih potrebujejo različni grafi, ter način, kako le-te poenotiti v modularni sistem. Kljub temu da je Ogre3D precej robusten pogon, se njegova zastarelost že kaže na nekaterih področjih. Aplikacija je zasnovana čim bolj modularno, zato da bi bilo dodajanje novih grafov scene čim bolj preprosto.

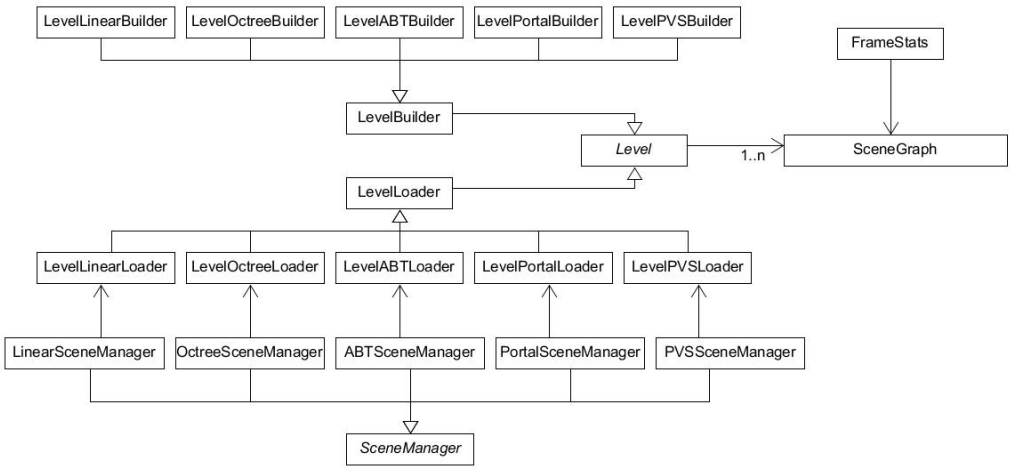
Osrednji del aplikacije predstavlja instanca razreda SceneGraph, ki je zgrajen kot načrtovalski vzorec Singleton [32]. Zaradi osrednje narave je v kodi aplikacije ta objekt dosegljiv kjerkoli. Preko tega objekta poteka večina komunikacije z Ogre3D pogonom in deluje kot lepilo med ostalimi deli aplikacije, kot so na primer:

uporabniški vmesnik, sistem nalagalnikov, razčlenjevalnikov in graditeljev. Preko vmesnika, ki ga izpostavlja SceneGraph instanca, so izvršene uporabnikove akcije na vmesniku ter registriranje nalagalnika ali graditelja za določen tip grafa scene, ipd.

Grafi scene so si tako različni, da zahtevajo drugačen način nalaganja oziroma grajenja. Za omogočanje takšne različnosti je proces nalaganja in grajenja grafov realiziran preko virtualnega abstraktnega vmesnika Level. Iz tega vmesnika sta izpeljana razreda LevelLoader in LevelBuilder, kjer je prvi namenjen za specializirano implementacijo nalagalnikov, drugi pa za implementacijo graditeljev grafov. Graditelj grafa scene prebere vsebino sveta iz pogona Ogre3D, izvede algoritem za razdeljevanje vsebine v prostoru, primeren za določen tip grafa scene, ter izpiše rezultat v datoteko. Octree graditelj, na primer, za vsak objekt določi celico, kjer se objekt nahaja in zapiše to informacijo v datoteko. Nalagalnik pa zna to datoteko razčlenjevati in nalagati vsebino v pogon Ogre3D. Takšen sistem omogoča enkratno izgradnjo opisa sveta za nek tip grafa scene in večkratno uporabo. Prav tako lahko na preprost način zgradimo optimalen format opisa sveta za čisto vsak tip grafa scene.

Nalagalniki so namenjeni zgolj razčlenjevanju datotek v določenem formatu in ne služijo za dejansko implementacijo grafa scene. Pogon Ogre3D ima funkcionalnost grafov scene, ki je implementirana preko vtičnikov, ki se lahko v realnem času ustvarijo ali uničijo. Vsak specializiran nalagalnik mora vzpostaviti ustrezen vtičnik, na primer PVSSceneManager, preko katerega se ustvari svet, zapisan v datoteki, ki jo razčlenjuje. Pogon bo nato avtomatsko klical funkcije za posodabljanje in upodabljanje. Razredni diagram testne aplikacije in povezovanje z abstraktnim vmesnikom SceneManager sta prikazana na sliki 3.8. Osrednji objekt aplikacije vsebuje tudi razred FrameStats, ki je namenjen izpisu in zajemu podatkov.

Uporabniški vmesnik je zgrajen s pomočjo CEGUI (Crazy Eddie's Graphics User Interface) [15] knjižnice, ki je tudi ena izmed priporočljivih knjižnic za izgradnjo uporabniškega vmesnika v Ogre3D. Knjižnica omogoča dva načina opisovanja vmesnika, in sicer direktno preko programske kode, ali preko posebnih datotek s končnico .layout, pri čemer je prvi način zelo okoren, saj vsaka sprememba zahteva ponovno prevajanje in povezovanje aplikacije. Zaradi očitnih prednosti, ki jih ponuja, testna aplikacija uporablja za vsa okna drugi način. Vsaka implementa-



Slika 3.8: Poenostavljen UML diagram testne aplikacije.

cija graditelja lahko ustvari in registrira posebno okno, kjer so tipične možnosti za izgradnjo določenega tipa grafa. To okno se prikaže preko Option menija, potem ko uporabnik izbere ustrezen graf scene v meniju Scene. Na takšen način je tudi implementacija vmesnika za možnosti grafov neodvisna od ostalega dela aplikacije. Nenazadnje je potrebno še omeniti, da uporabniški vmesnik vedno komunicira le z globalno instanco SceneGraph, kar pomeni, da se z drugačnim vmesnikom le-ta lahko kadarkoli spremeni.

Na sliki 3.9 je predstavljen uporabniški vmesnik testne aplikacije. Menija File in Manager sta namenjena izbiri sveta in grafa scene, medtem ko se v meniju Scene nahajajo opcije, specifične za izbran svet. Izbira lastnosti za neki graf scene je možna preko menija Option, ki glede na izbran graf scene vsebuje nastavljive opcije. Primer takšnega okna opcij je prav tako prikazan na prej omenjeni sliki. Naslednja dva menija sta Run in Build, prvi je namenjen za zagon izbranega grafa scene na določenem svetu, drugi pa je namenjen le grajenju grafa scene. Z izbiro menija Camera, lahko prestavljamo kamero v svetu na vnaprej določene položaje, ki so nastavljivi preko Maye. Zadnja menija pa sta namenjena evaluaciji scenarijev. Meni Tour popelje kamero po krivulji, ki je zgrajena v Mayi in je bila izvožena skupaj s svetom. V katerem koli trenutku scenarija ima uporabnik možnost izvoza vseh metrik, ki so se zbirale med pomikanjem kamere po krivulji z izbiro menija Stats.

Slika 3.9: Grafični vmesnik testne aplikacije.

```
<?xml version="1.0" encoding="UTF-8"?>
<Manager>
  <Linear/>
  <Octree SizeX="350" SizeY="61" SizeZ="350" Depth="4"/>
  <Portal/>
  <PVS SizeX="351" SizeY="61" SizeZ="351" CellCountX="10"
   CellCountY="10"  CellCountZ="10"/>
  <ABT Depth="4" PlaneCount="20"/>
</Manager>
```

Slika 3.10: Začetne nastavitve grafov scen v datoteki ManagerSettings.xml.

V veliki meri je testna aplikacija nastavljiva preko zunanjih datotek, zato da uporabniku ni potrebno prevajati aplikacije pri majhnih spremembah, kot sta na primer izbira sveta ali grafa scene ob zagonu. Izvorna vsebina pridobljena preko orodja OgreMax je shranjena ločeno od datotek, ki jih generira testna aplikacija. Vsebina za določen svet je opisana z zunanjo datoteko LevelSettings.xml,

ki določa lokacijo mrežnih modelov, senčilnikov in tekstur ter lokacijo datoteke ManagerSettings.xml, ki določa trenutne nastavitve grafov scen (slika 3.10). V slednji datoteki so določene predvsem začetne nastavitve grafov scene, ki opisujejo meje sveta, število delitev in tako dalje ter predstavljajo med drugim tudi začetne vrednosti v grafičnem vmesniku (slika 3.9).

Kljub temu, da Ogre3D velja, kot preverjen in stabilen pogon, velja omeniti težave, ki so se pojavile med njegovim razvojem. Zaradi cilja po čim bolj realističnem testiranju grafov je bila izbira 3D vmesnika DirectX11 nujna, saj je to najpogostejši 3D vmesnik za igre na osebnih računalnikih. Izkazalo se je, da podpora za DirectX11 ni ravno najboljša in da pogon vsebuje par napak pri povezavi CEGUI in Ogre3D. Drugo težavo predstavljajo senčilniki, ki jih mora pri uporabi DirectX11 uporabnik napisati sam. Ogre3D v osnovi uporablja skriptni jezik za definicijo materialov, ki pa žal ni več delujoč z novo verzijo DirectX.

Poglavje 4

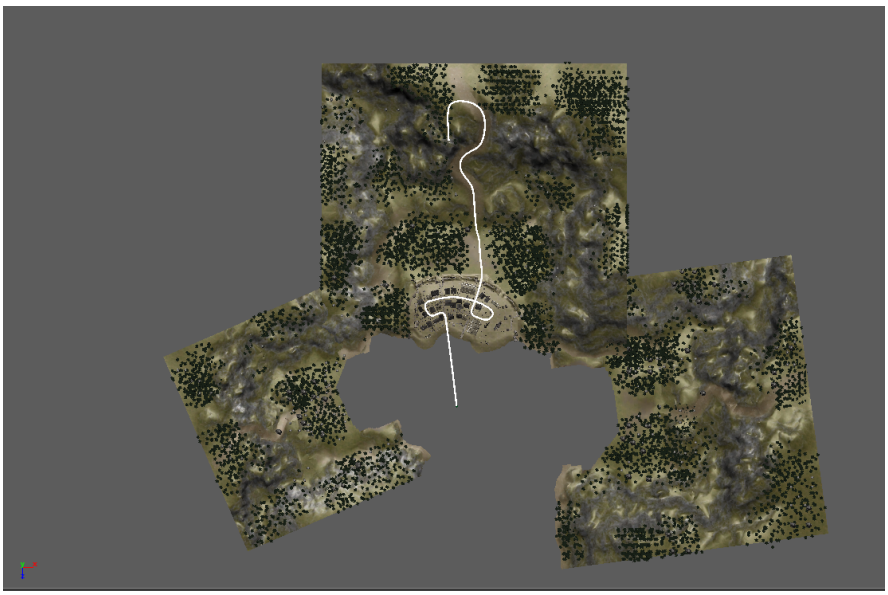
Scenariji testiranja

4.1 Odprti svet

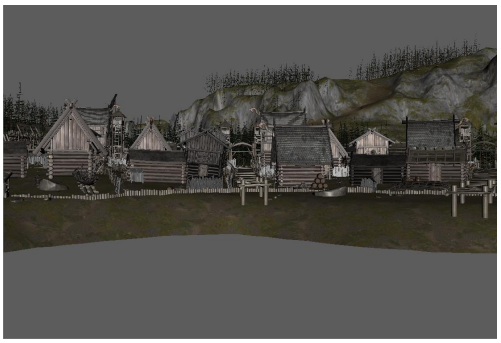
Unity Asset Store ponuja svet z imenom Viking Village, ki ima ustrezne lastnosti odprtega sveta. Svet vsebuje v središču majhno vikinško vas, ki je obdana z veliko terena. Teren je v središču sveta prekrit z ravninami, medtem ko je ob robovih precej hribovit. Vas je sestavljena iz raznih hiš, upodobljenih v realistični velikosti in je prepletena s koridorji med hišami. To nam omogoči, da lahko popeljemo igralca skozi svet, kjer bo velik del sveta povsem zakrit, na primer znotraj vasi, vse do položajev na robovih terena, kjer bo viden celoten svet. Svetu so bili naknadno dodani še drugi objekti, predvsem drevesa in kamni, saj v osnovi teren okoli vasi ni zapolnjen z objekti. Celotno število objektov, ki se nahaja v svetu, se vrti okoli številke 16.000.

Krivulja po kateri potuje kamera je bila izbrana z namenom približanja gibanju igralca skozi odpri svet (slika 4.1). Pot se začne s širokim pogledom na celoten svet (slika 4.2), kar bo zelo otežilo delovanje vseh različic grafov, saj je možnost odstranjevanja velikega števila objektov majhna. Pričakuje se najboljše delovanje grafov z binarnim razdeljevanjem sveta. Sledi potovanje znotraj vasi, ki se nahaja v središču sveta (slika 4.3). Tukaj je velik del sveta zakrit, čeprav se zaradi dejstva, da se vas nahaja v središču sveta, pričakuje slabše rezultate pri binarnemu razdeljevanju prostora in boljše pri grafih kot so npr. portali in PVS. Sledi izhod iz vasi (slika 4.4) oziroma iz središča proti robovom terena, kjer se bo delovanje

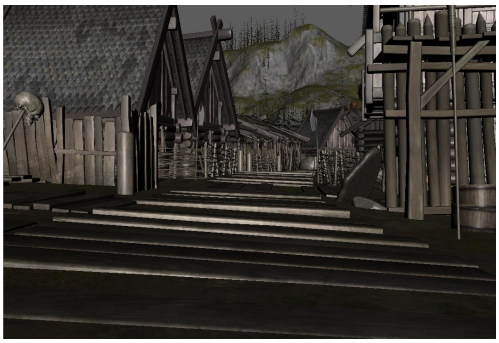
vseh grafov občutno izboljšalo, saj kamera potuje stran in je hkrati obrnjena stran od središča sveta. Konec poti predstavlja pogled na celoten svet, kjer se izmeri največja možna obremenitev vseh grafov (slika 4.5). Celotna pot je dolga 169 sekund.



Slika 4.1: Potek scenarija v odprtem svetu.

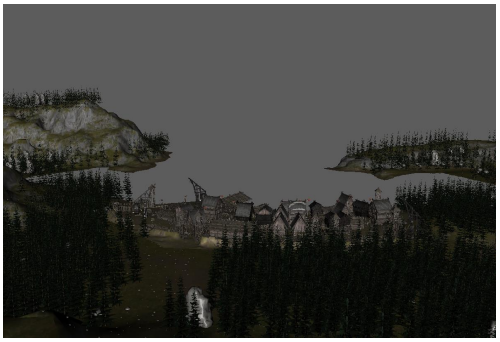


Slika 4.2: Začetni pogled na vas.



Slika 4.3: Pogled znotraj vasi.

Slika 4.4: Izhod iz vasi.

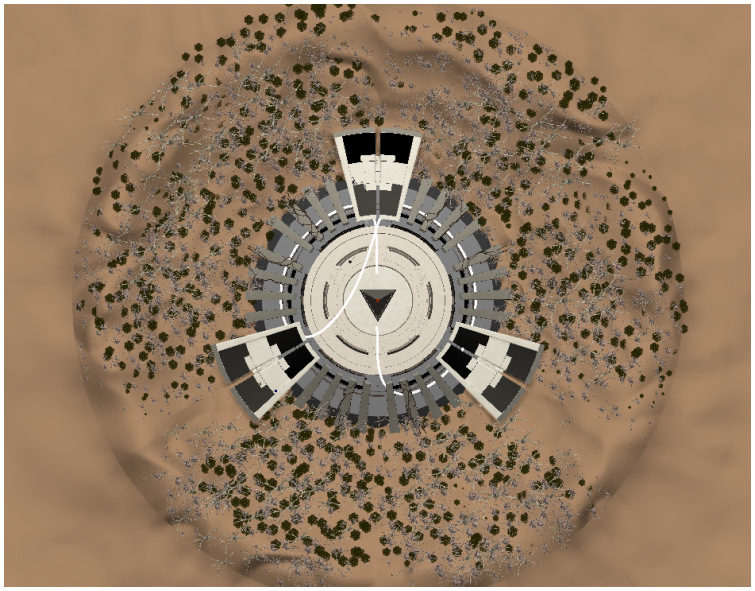


Slika 4.5: Končni pogled na vas.

4.2 Pol-odprti svet

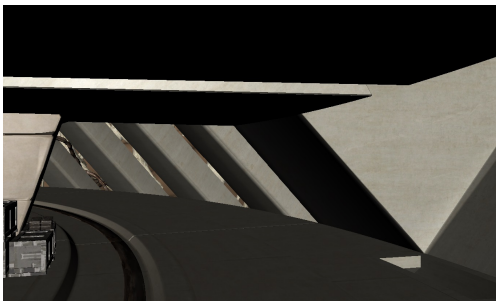
Svet z imenom Courtyard iz Unity Asset Store se najbolje približa pol-odprtemu svetu. V tem primeru je pot igralca omejena znotraj zgradbe, iz katere igralec nima nikakršnih izhodov. Zgradba je sestavljena iz osrednjega prostora, kjer se nahaja piramida in prehodov v zunanji koridor (slika 4.6). V temu koridorju so velike odprtine, skozi katere lahko igralec vidi zunanji svet oziroma teren. Okoli koridorja so enakomerno postavljeni trije stolpi, ki igralcu dovoljujejo, da vstopi vanje. Večina terena ni vidnega, razen pri potovanju skozi zunanji koridor, zato se le tu pojavljajo objekti na terenu, kot so na primer drevesa, kamenje in hlodi. Tudi v temu primeru je bil prvotni svet obogaten z dodatnimi objekti, saj je bil teren prazen. Tudi v tem primeru je število vseh objektov v svetu okoli 16.000.

Pot kamere se začne znotraj zaključenega prostora, kjer se najprej naredi prehod iz osrednjega prostora na zunanji obroč (slika 4.7). V temu delu poti ima igralec pogled na zunanji svet, kjer so vsi grafi scene najbolj obremenjeni in ga dvakrat prekine stolp (slika 4.8). Sledi zavoj v središče sveta s prehodom na drugo stran zunanjega obroča. Tukaj se pogled na zunanji svet še enkrat ponovi. Pot se konča s ponovnim zavojem k središču sveta, proti piramidi (slika 4.9). Za razliko od odprtega tipa sveta lahko tukaj pričakujemo dobro konsistenco in performančnost upodabljanja grafov tipa portal in PVS, medtem ko bodo binarne porazdelitve imele načeloma slabše performance. Celotna dolžina scenarija je 83 sekund.

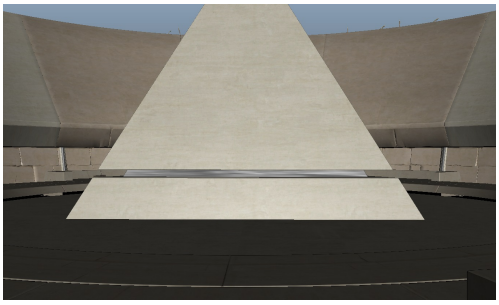


Slika 4.6: Potek scenarija v pol-odprtem svetu.

Slika 4.7: Prehod iz osrednjega prostora.



Slika 4.8: Koridor za zunanjem obroču.

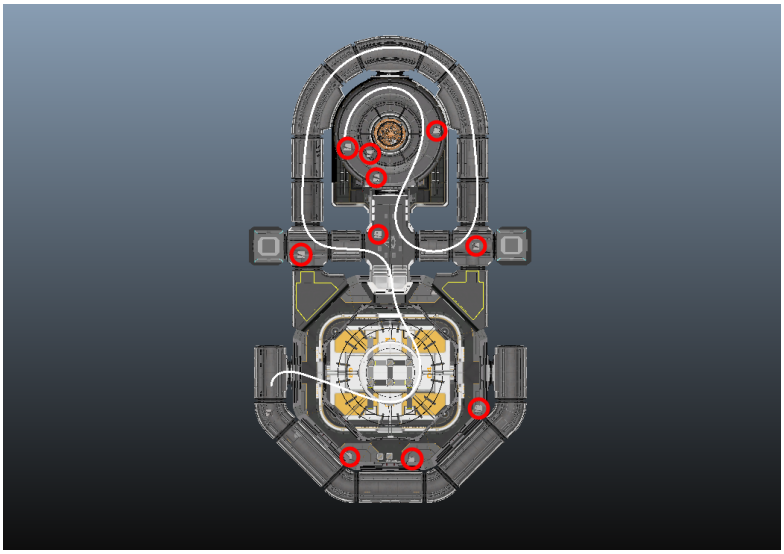


Slika 4.9: Končni pogled na piramido.

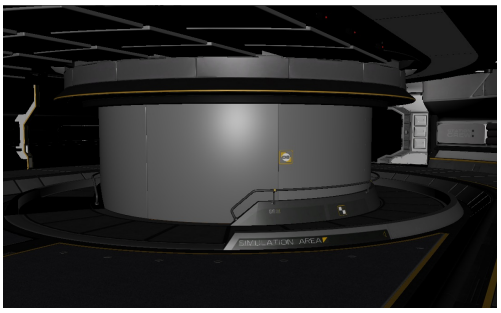
4.3 Zaprti svet

Zaprtemu svetu zelo dobro ustreza svet imenovan UnityLab iz Unity Asset Store. Svet predstavlja futuristično laboratorij, ki je sestavljenih iz dveh večjih prostorov, kjer je večina objektov vidnih, in iz koridorjev, ki povezujejo oba prostora. Edina prostora, kjer je možno rahlo svobodnejše gibanje sta imenovana Telovadnica in Poslušalnica, ki sta sestavljena iz dveh nivojev. V primerjavi s prejšnimi svetovi je gibanje zelo omejeno. Celotno število objektov, ki se nahajajo v svetu, je okoli 15.000. Ker je v osnovni obliki svet vseboval premajhno količino objektov, je bil le-ta dopolnjen z dodatnimi objekti v obliki palete kock. Vsaka kocka vsebuje po 1.000 objektov, s katerimi zelo otežimo delovanje grafov. Na (slika 4.10) je z rdečo barvo označena lokacija palet, medtem ko je pot kamere označena z belo barvo.

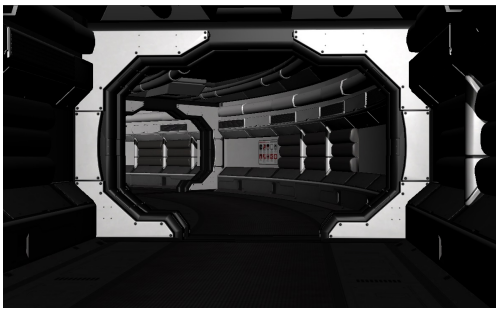
Pot igralca se začne v odprtem prostoru imenovem Poslušalnica, kjer kamera naredi kratek obhod (slika 4.11). Kamera nato preide v koridor, ki vodi okoli Poslušalnice (slika 4.12). Ko se obhod konča, je kamera na v prvotnem položaju, ki ga je imela ob izhodu iz prvega prostora. Sledi prehod v drugi večji prostor imenovan Telovadnica (slika 4.13), od koder se kamera popelje po prostoru v novi koridor, kjer se scenarij konča. Ker je prostor zaprte narave, lahko pričakujemo zelo dobro performančno delovanje in konsistenco upodabljanja sistema portalov ter PVS grafa scene. Algoritmi binarnega razdeljevanja prostora se bodo odrezali slabše, vendar še vedno bolje kot linearen seznam. Celotna pot je dolga 83 sekund.



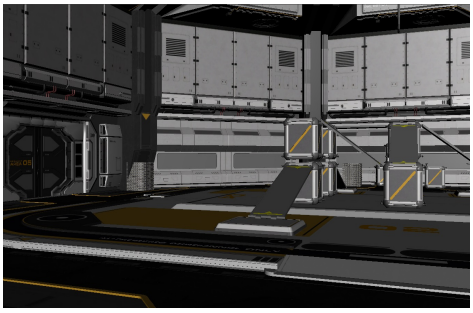
Slika 4.10: Potek scenarija v zaprtem svetu.



Slika 4.11: Začetek scenarija v zaprtem svetu.



Slika 4.12: Prehod v koridor v zaprtem svetu.



Slika 4.13: Pogled na telovadnico v zaprtem svetu.

Poglavje 5

Rezultati

5.1 Uvod

V temu poglavju so navedene meritve, ki so bile izvedene na opisanih grafih scen s pomočjo testne aplikacije. Aplikacija vzorči vsako sekundo spodaj uporabljene metrike. Vse meritve so bile izvedene na računalniku, ki se pri trenutnem stanju tehnologije uvršča v nižji srednji razred. Spodaj so navedene strojne in programske značilnosti okolja:

CPU: Intel Core i7 920, 2.76 GHz

Grafična kartica: AMD Sapphire 5850, 1GB VRAM

Matična plošča: ASUS P6X58D-E

RAM: Kingston 1066Mhz, 6GB

Disk: Western Digital WD1002FAEX, 1TB

Operacijski sistem: Windows 8.1 64bit

Različica pogona Ogre: 1.9 64bit

3D API: DirectX 11

5.2 Metrike

Učinkovitost grafov scen lahko izmerimo na več načinov, vse pa je odvisno od tega, katera metrika nas zanima. Čeprav se kot merilo za učinkovitost najpogosteje uporablja hitrost, so prav tako pomembne še druge, na prvi pogled manj pomembne lastnosti.

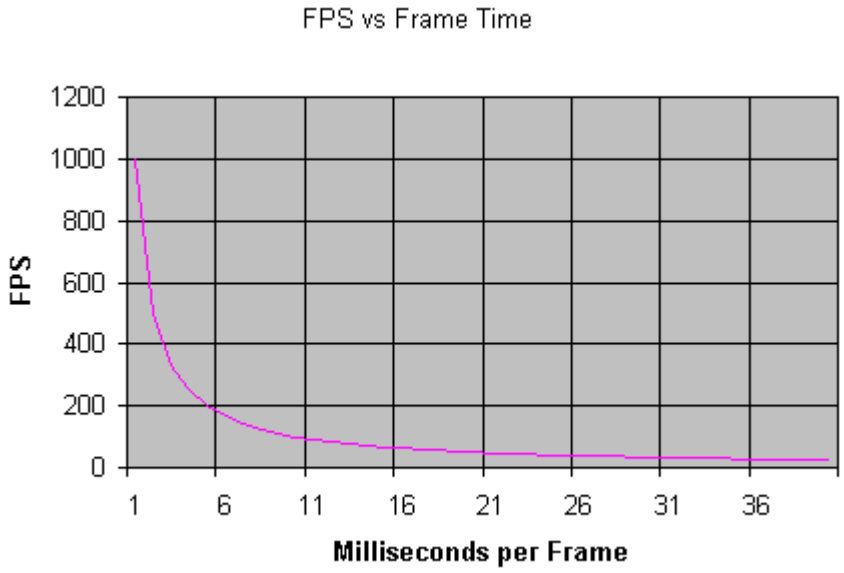
V nadaljevanju so predstavljene nekatere izmed možnih metrik ocenjevanja grafov scen. Pri testiranju grafov scen so bile izbrane metrike predvsem z vidika ocenjevanja učinkovitosti odstranjevanja in posledično performančnosti. Metrike, ki najbolje opisujejo iskane rezultate so mspf (angl. miliseconds per frame – milisekunde na vzorec), število odstranjenih objektov in število vidnih celic. Preostale metrike so bile izločene zaradi različnih razlogov. Zaradi slabe zanesljivosti metrika fps ni bila uporabljena. Čas za odstranjevanje je odvisen od same implementacije in je že vsebovan v metriki mspf. Metrika števila upodobljenih objektov je posredno razvidna iz števila odstranitev, medtem ko je čas izgradnje grafa odvisen od implementacije.

5.2.1 Mspf

Oznaka mspf označuje v milisekundah čas med dvema vzorcema (angl. frame) in je najbolj verodostojna metrika za merjenje performančnosti. Čeprav se najpogosteje govori o metriki fps, je ta zelo zavajajoča. Povezava med dvema se zapiše z enačbo mspf = 1 / fps. Recimo torej, da imamo tri meritve za fps: 900, 450 in 300. Na podlagi prej omenjene enačbe je čas med vzorci 1,111 ms, 2,222 ms in 3,333 ms. Torej sprememba od 900 fps do 450 fps ima učinek povečanja časa med vzorci, in sicer za 1,111 ms, toda enak rezultat je pridobljen tudi pri spremembi od 450 fps do 300 fps. Še ekstremnejši primer predstavlja padec od 60 fps do 56,25 fps, ki se ravno tako odraža v razliki 1,111ms. Problem metrike fps je v njeni nelinearnosti in dejstvu, da ni najbolj primerna za merjenje performančnosti, kar lahko vidimo na sliki 5.1 [7].

5.2.2 Fps

Kljub svoji nelinearnosti pa fps metrika ni povsem neuporabna. Odlika te metrike je predvsem v razumljivosti in razširjenosti. Fps metriko lahko še vedno uporabimo



Slika 5.1: Graf primera razmerja mspf in fps.

za hitro detekcijo minimumov in maksimumov.

5.2.3 Čas za odstranjevanje na nivoju grafa

Omenjena metrika podaja čas v milisekundah, ki so potrebne za odstranjevanje na nivoju grafa. Odstranjevanje s pomočjo grafa izkorišča prostorsko razdeljevanje prostora, ki je specifično za vsak graf posebej. Z vidnostjo teh področij se določi tudi vidnost objektov, ki se nahajajo znotraj prostora.

5.2.4 Število odstranjenih objektov

Namen vseh grafov scen je odstranjevanje velike količine objektov, za katere se predvideva, da niso vidni. Odstranjevanje poteka na dveh nivojih, in sicer na nivoju samega grafa, kjer se odstranjujejo zaključene skupine objektov, ter na nivoju kamere, kjer se vidnost ugotavlja za vsak objekt posebej.

5.2.5 Število vidnih celic

Uporablja se samo pri ABT in Octree drevesu. Z njim se določa število celic, ki so vidne v določenem vzorcu. Na takšen način se posredno vpliva tudi na čas, ki je potreben za odstranjevanje objektov v teh celicah.

5.2.6 Število upodobljenih objektov

Dejansko število upodobljenih objektov se med grafi scen lahko precej razlikuje. To odstopanje vpliva na čas upodabljanja in, v določenih primerih, celo na natančnost upodabljanja.

5.2.7 Čas izgradnje sveta

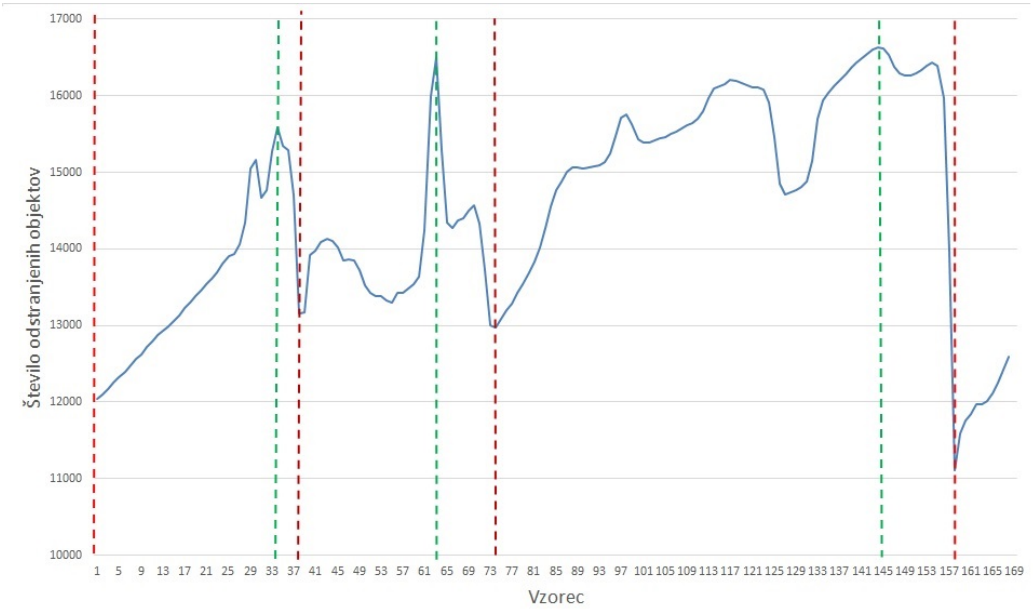
Omenjena metrika je zelo pomembna za oceno splošne uporabnost grafa scene, saj čas, ki je potreben za grajenje posredno določa tudi čas, ki je potreben za uvajanje sprememb v svet. Slednje je predvsem pomembno za končne uporabnike 3D pogona, kot so na primer: oblikovalci svetov, modelarji, animatorji in oblikovalci iger.

5.3 Odprti svet

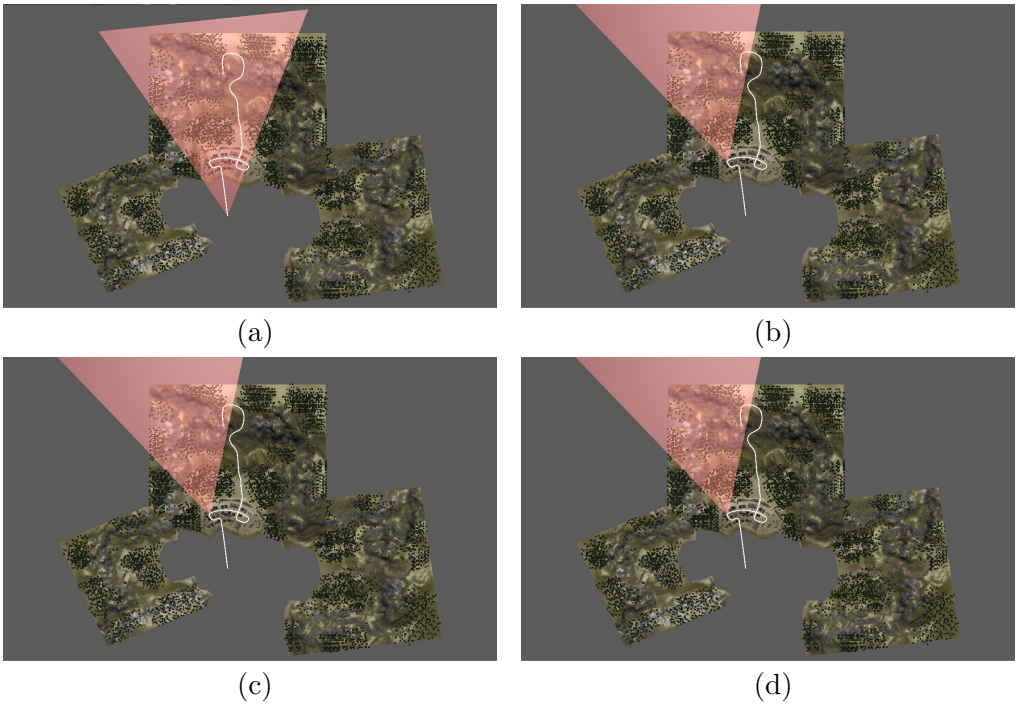
5.3.1 Linearen seznam

5.3.1.1 Odstranjevanje objektov

Enostavno odstranjevanje odstrani s kamero precejšni del objektov odprtega sveta. Število odstranjenih objektov se giblje med 11.000 in 16.500, s povprečjem 12.000 objektov. Na sliki 5.2 je prikazano v odvisnosti od vzorca število odstranjenih objektov. Tekom potovanja kamere število doseže več minimumov, in sicer pri vzorcih 1, 38, 74 in 158. Pri vzorcu 1, na sliki 5.3a, je kamera na začetni poziciji z zelo širokim pogledom na vas in teren za njo, ki vsebuje velik del objektov v svetu. Podobno velja za vzorca 38 in 74, kjer je pogled prav tako usmerjen na teren za vasjo (sliki 5.3b in 5.3c). Od vzorca 74 dalje kamera potuje izven vasi, vedno bolj proti robovom sveta, kar je razvidno iz postopnega višanja števila odstranitev. Najnižji padec števila odstranjenih objektov je zabeležen pri vzorcu 158 (slika 5.3d), kjer je v vidnem polju kamere zajet skoraj celoten svet.



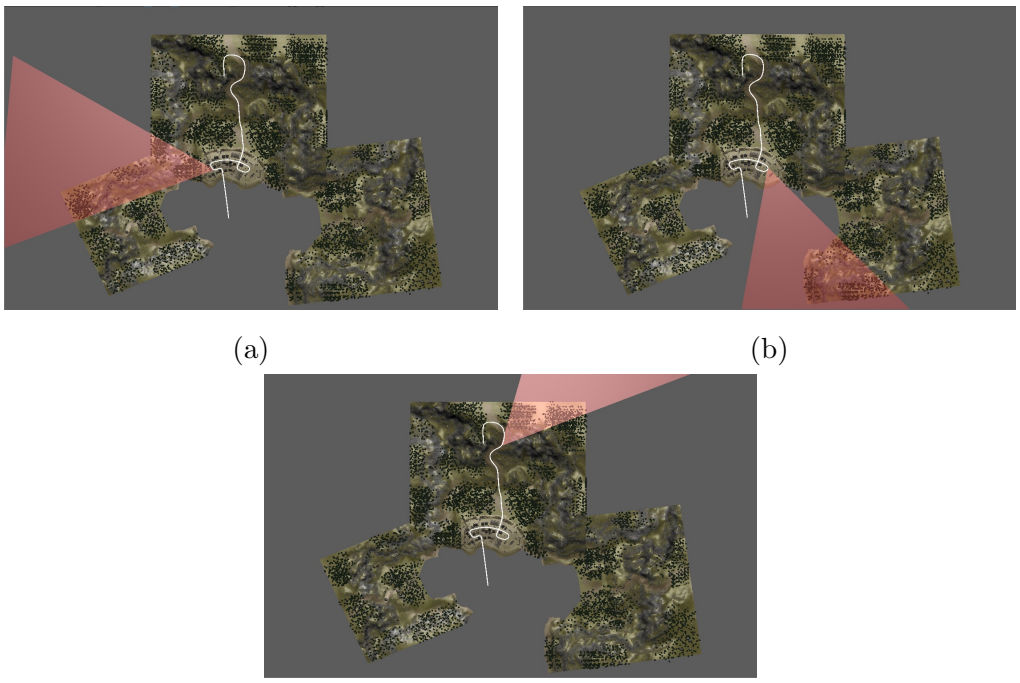
Slika 5.2: Število odstranjenih objektov z linearnim seznamom v odprtem svetu.



(a) (b) (c) (d)

Slika 5.3: Minimumi števila odstranjenih objektov pri linearnem seznamu v odprtem svetu.

Število odstranitev doživi tudi več maksimumov, in sicer pri vzorcih 33, 63 ter 145 (slike 5.4a, 5.4b in 5.4c). Vsem trem vzorcem je skupno vidno polje kamere, ki ne zajema vasi in večine terena za njo. Največje število odstranitev je pri vzorcu 63, ko ima kamera pogled proti morju, s čimer se odstranijo skoraj vsi objekti v svetu.



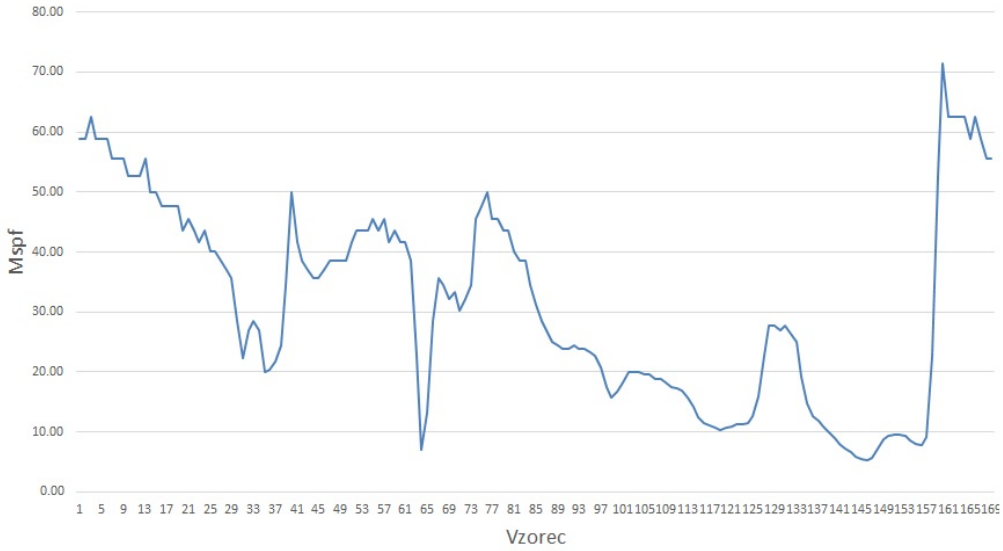
(a) (b)

(c)

Slika 5.4: Maksimumi števila odstranjenih objektov pri linearnem seznamu v odprtem svetu.

5.3.1.2 Performančnost

Izmed vseh grafov scene je pri odprtem svetu število odstranjenih objektov na nivoju kamere največje. Primerjava odstranjevanja in performančnosti (slika 5.5) nakazuje na pozitivno linearno zvezo. Pri maksimumih odstranjevanja je namreč hitrost največja in obratno.



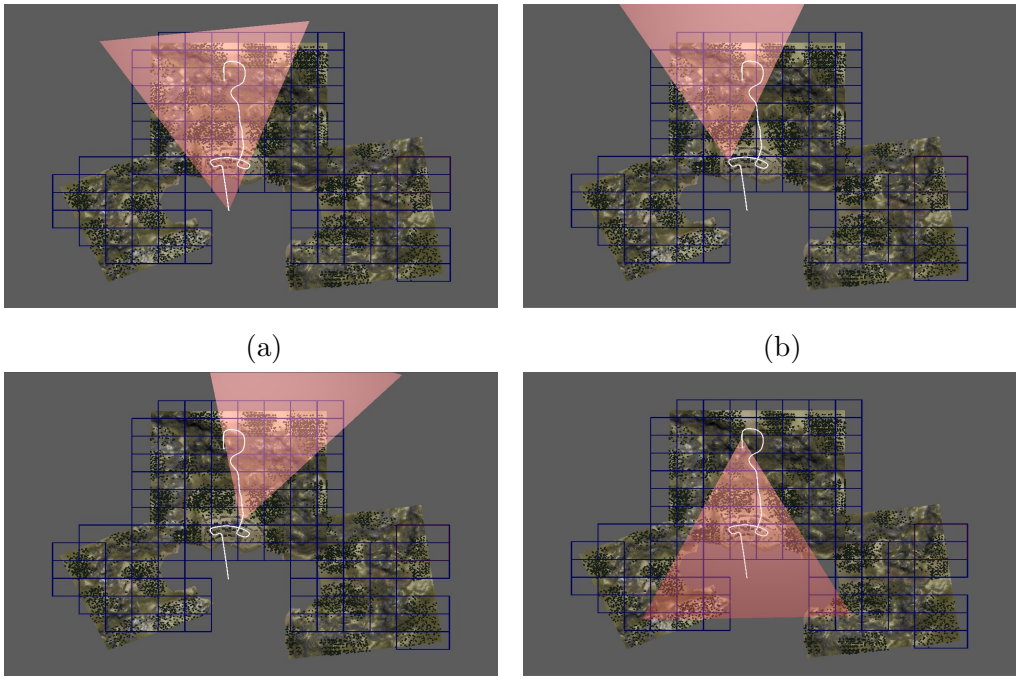
Slika 5.5: Mspf performančnost linearnega seznama v odprtem svetu.

5.3.2 Octree

5.3.2.1 Odstranjevanje objektov

Rezultati odstranjevanja različnih globin deljenja grafa si sledijo po podobnem vzorcu in se razlikujejo kvečjemu po magnitudi. Globalni minimum je dosežen pri vzorcu 162 (slika 5.6d), ko pogled kamere zajame vas in večji del sveta. Preostali minimumi se pojavijo pri vzorcih 1, 39 ter 72. V prvem primeru je pogled zopet usmerjen na vas in okolico, vendar ni tako širok kot na koncu scenarija (slika 5.6a). Pri vzorcih 39 (slika 5.6b) in 72 (slika 5.6c) pa se pogled kamere obrača proti terenu za vasjo, ki vsebuje veliko objektov. Najboljši rezultat odstranjevanja na nivoju grafa je zabeležen pri vzorcu 144 (slika 5.7c), kjer se kamera nahaja na robu in je obrnjena stran od središča sveta. Pri vzorcih 35 (slika 5.7a) in 62 (slika 5.7b) pa je kamera obrnjena stran od središča vasi in ne proti delom terena z najmanj objekti.

Najslabši rezultati odstranjevanja so pri globini 1 ter na koncu scenarija. Večino poti se kamera nahaja na presečišču oktantov, kar onemogoči kakršnokoli odstranjevanje. Slika 5.8 prikazuje oktante Octree drevesa in z belo barvo označeno pot, ki jo opravi kamera. Število vidnih oktantov (slika 5.11) ostaja ves čas sce-



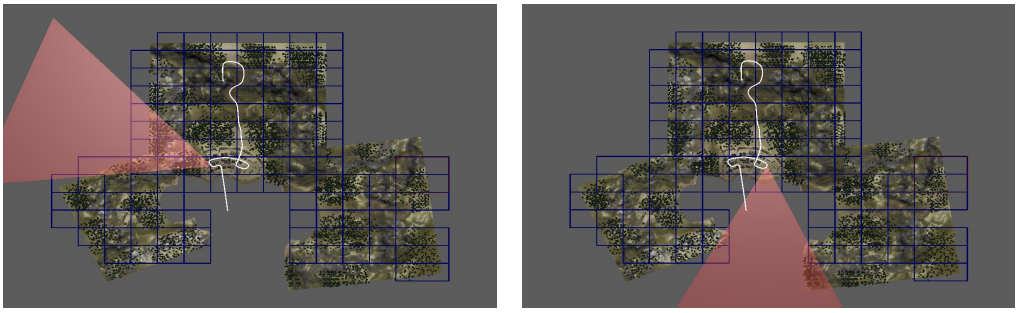
(a) (b)

(c) (d)

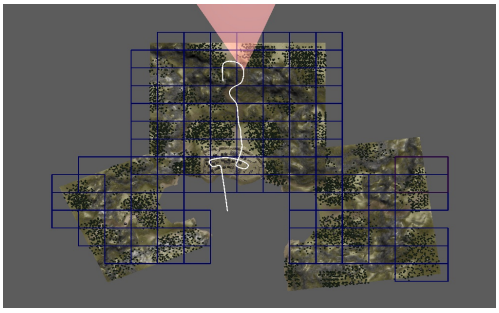
Slika 5.6: Mimimumi števila odstranjenih objektov pri ohlapnem Octree v odprtem svetu.

narija skoraj nespremenjeno in se giblje okoli 9, kar predstavlja celotno število oktantov Octree drevesa globine 1. Na koncu scenarija se pojavita kratki okni (vzorci 138-145 in 152-155), kjer kamera po y osi prehaja iz nižjega v višji nivo ter se premika izven presečišča oktantov, kar omogoči edino odstranjevanje na nivoju grafa.

Z večanjem globine grafa se povečuje količina odstranjenih objektov na nivoju grafa, vendar le do določene mere. Najboljši rezultati so zabeleženi pri globinah 6 in 8, kjer je količina odstranitev približno enaka (slika 5.10). Povečevanje globine grafa eksponentno poveča število vidnih oktantov (slika 5.11). To pa se ne odraža v isti meri pri količini odstranitev, saj se le-te ustalijo pri globini 8. Razlog za takšno obnašanje je majhna zapolnjenost oktantov pri globinah 6, 7 in 8. Na sliki 5.9 je vidna gostota oktantov, kjer rjava barva predstavlja oktante globine 8. Največja koncentracija je znotraj vasi in na terenu okoli nje.

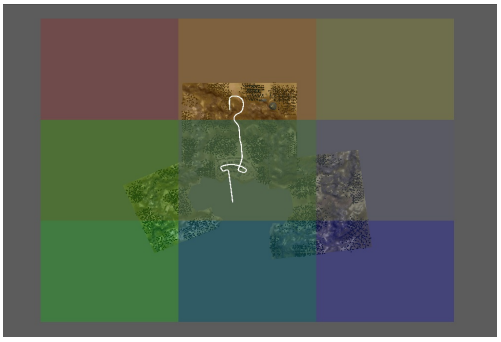


(a) (b)

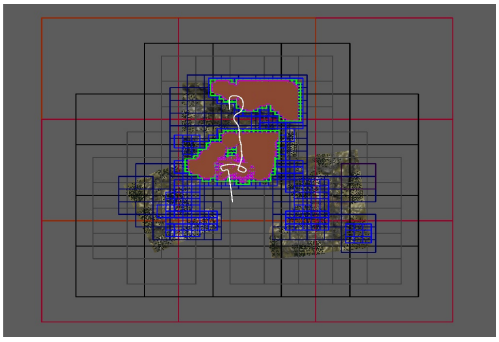


(c)

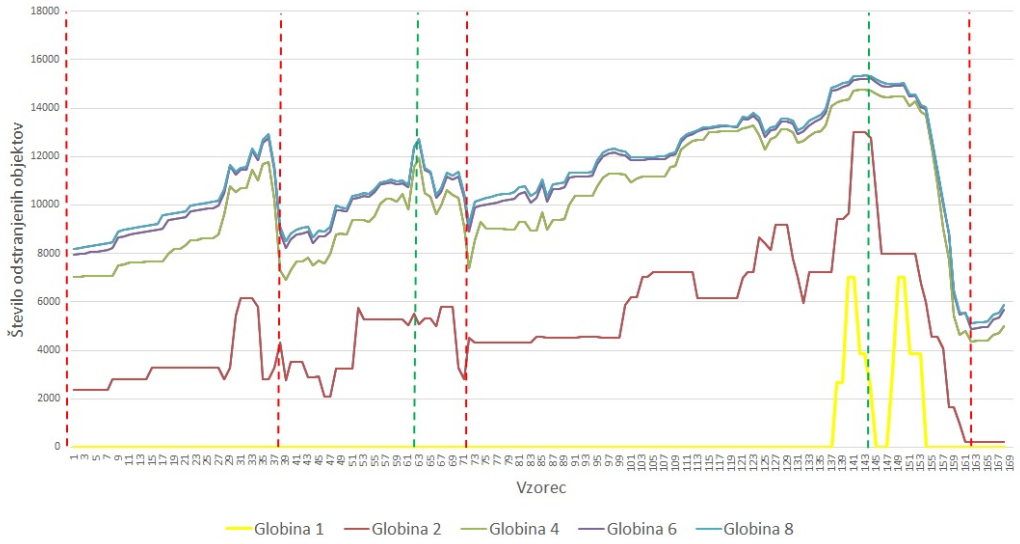
Slika 5.7: Maksimumi števila odstranjenih objektov pri ohlapnem Octree v odprtem svetu.



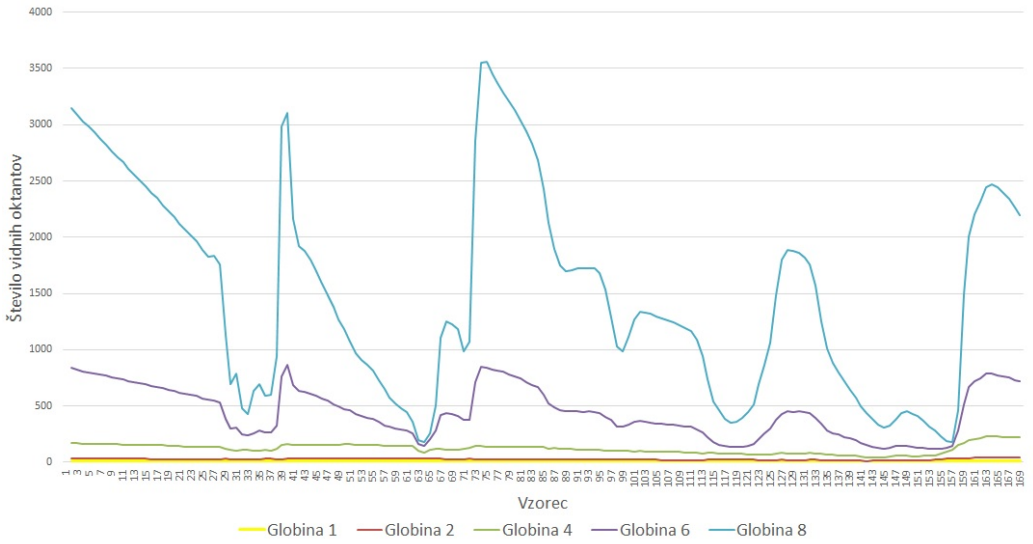
Slika 5.8: Oktanti ohlapnega Octree drevesa globine 1 pri odprtem svetu.



Slika 5.9: Oktanti ohlapnega Octree drevesa pri globinah 6, 7 in 8.



Slika 5.10: Število odstranjenih objektov z Octree drevesom v odprtem svetu.

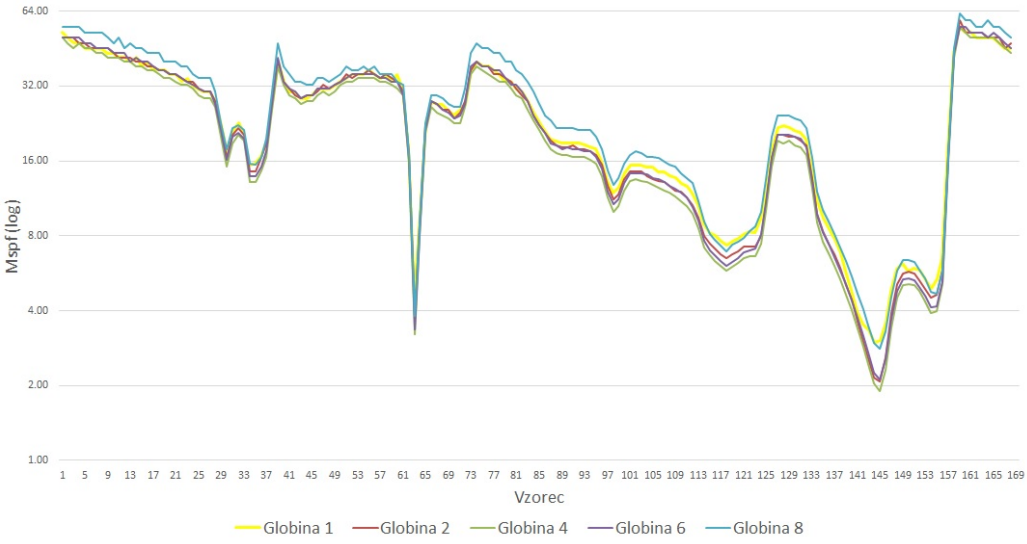


Slika 5.11: Število vidnih oktantov Octree drevesa v odprtem svetu.

5.3.2.2 Performančnost

Na sliki 5.12 so v logaritmični skali predstavljene performančne meritve grafov vseh globin. Najboljša različica Octree drevesa je globina 4, medtem ko predstavlja globina 8 najslabšo različico. Preveliko povečanje globine istočasno zelo poveča čas,

ki je potreben za odstranjevanje na nivoju grafa. Posledično so rezultati slabši, kot če bi uporabili minimalno razdeljevanje prostora, na primer pri globini 1. Pred ostalimi ima globina 4 prednost ves čas izvajanja scenarija, najbolj pa takrat, ko je število odstranitev največje (slika 5.10). V teh vzorcih je najboljša razdelitev odstranjevanja med grafom in kamero, ki je performančno najbolj ugodna. Globina 8 kljub temu, da bolje razdrobi svet, uvede preveliko število novih oktantov, da bi se to dejansko splačalo.



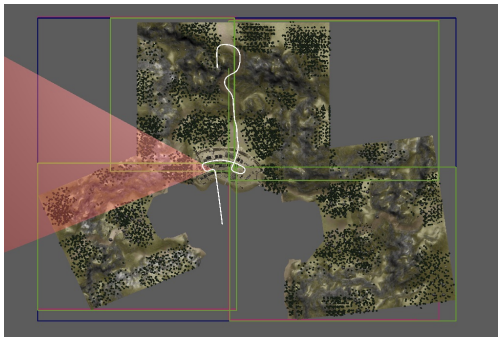
Slika 5.12: Mspf performančnost Octree drevesa v odprtem svetu.

5.3.3 ABT

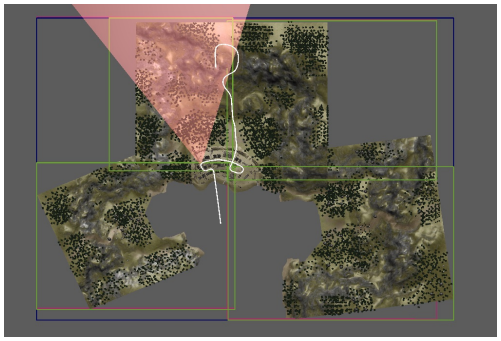
5.3.3.1 Odstranjevanje objektov

ABT prav tako sledi vzorcu povečevanja odstranitev na nivoju grafa s približevanjem vasi, kjer doseže lokalni maksimum pri vzorcu 34 (slika 5.13). Na tem položaju dobršen del ABT področij ni viden, saj kamera gleda stran od središča sveta. Pri vzorcu 38 (slika 5.14) se kot kamere spremeni nazaj proti središču, kar povzroči, podobno kot pri drevesu Octree, padec. Naslednji skok je pri vzorcu 63 (slika 5.15), kjer vidno polje zajame majhen del sveta. Potovanje izven vasi povečuje odstranitve do maksimuma, ki je zabeležen pri vzorcu 154 (slika 5.16), kjer se kamera nahaja zelo blizu roba sveta. Podobno kot pri Octree je globalni minimum

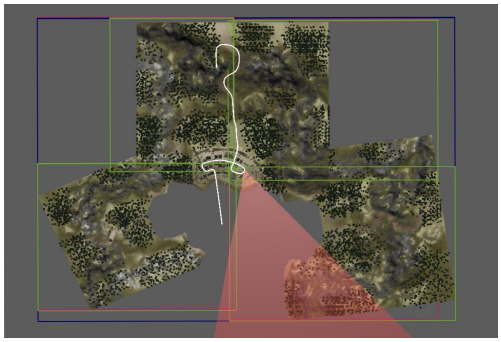
dosežen pri vzorcu 160 (slika 5.17), kjer kamera gleda na skoraj celoten svet. Pri nižjih delitvah ABT razdeli odprti svet na zelo regularna področja, ki so precej podobna Octree oktantom. Razlike začnejo izstopati šele pri višji deljenjih. Razlog je precejšna regularnost izbranega sveta. Vrednosti odstranjevanja tekom izvajanja scenarija so prikazane na sliki 5.18. Največje odstranitve so pri 5-kratni delitvi in najslabše pri enkratni delitvi. Višanje delitev hkrati poveča tudi število vidnih področij (slika 5.19), kar pa od delitve 4 dalje ne prispeva bistveno k večanju odstranitev.



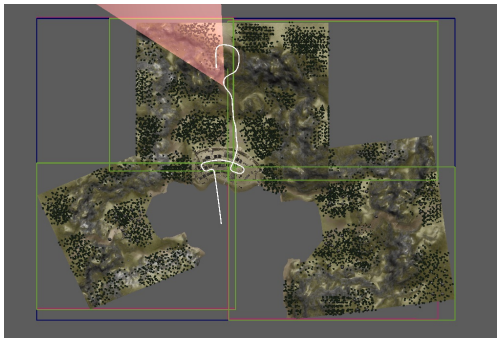
Slika 5.13: Vzorec 34 pri ABT drevesu v odprtem svetu.



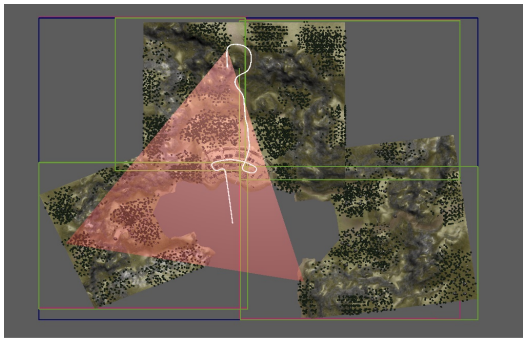
Slika 5.14: Vzorec 38 pri ABT drevesu v odprtem svetu.



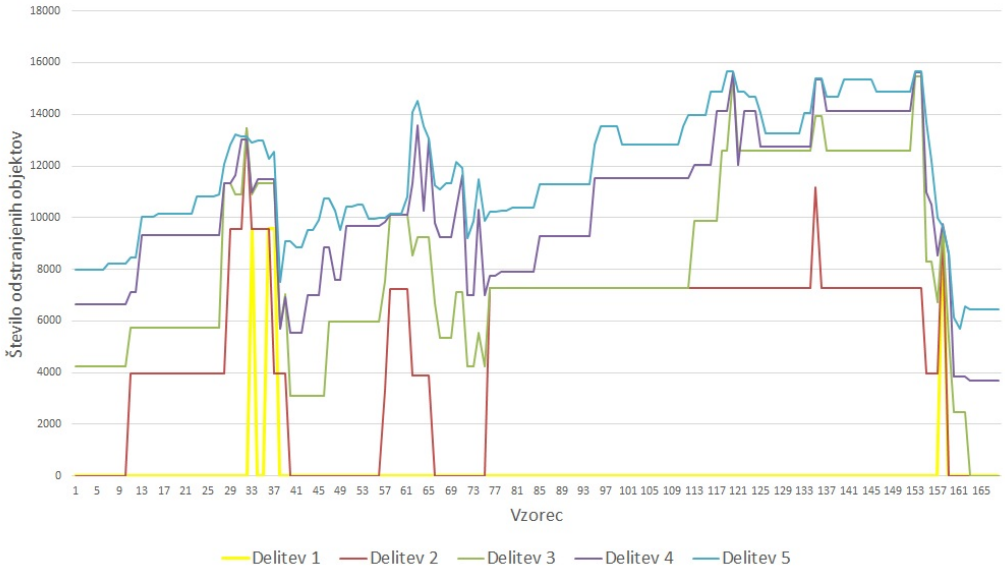
Slika 5.15: Vzorec 63 pri ABT drevesu v odprtem svetu.



Slika 5.16: Vzorec 154 pri ABT drevesu v odprtem svetu.



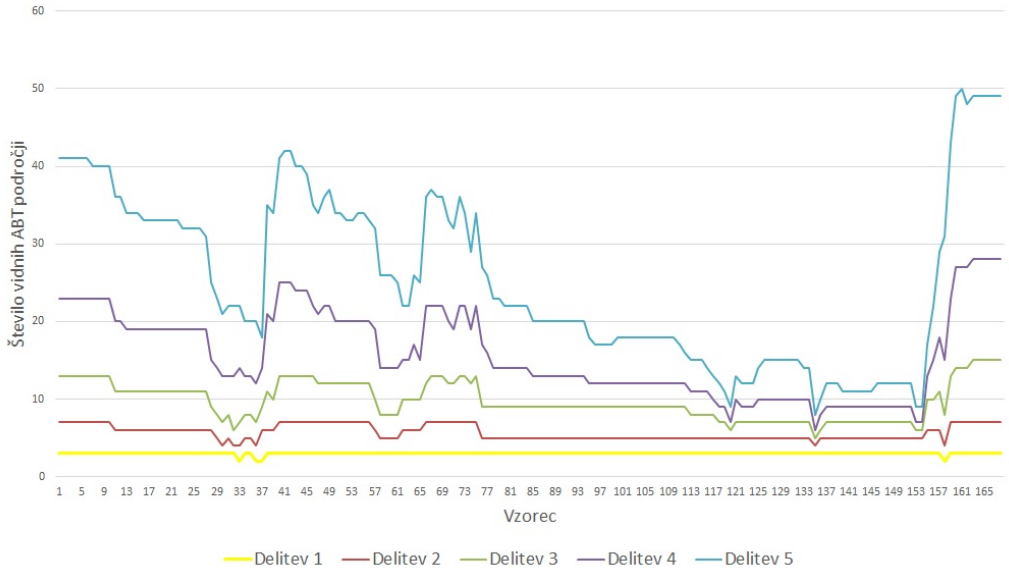
Slika 5.17: Vzorec 160 pri ABT drevesu v odprtem svetu.



Slika 5.18: Število odstranjenih objektov z ABT drevesom v odprtem svetu.

5.3.3.2 Performančnost

Najboljše rezultate je moč pridobiti s 5-kratno delitvijo, ki uspe kljub povečanju števila ABT področij (slika 5.20) le-ta zapolniti zgolj do tolikšne mere, da njihova obdelava ne vpliva negativno na hitrost. Najbolj so opazna odstopanja pri vzorcih, kjer je velik del ABT področij odstranjen, na primer vzorca 144 in 63. Čeprav so na prvi pogled vse globine dokaj slične, lahko znaša dejansko odstopanje od najboljši rezultatov do okoli 2 milisekunde, kar predstavlja predvsem pri večjih obremenitvah velik napredek v performančnosti.



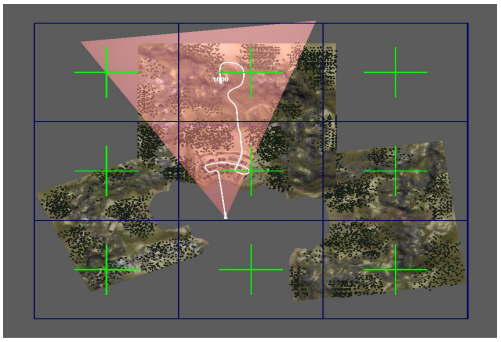
Slika 5.19: Število vidnih področij ABT drevesa v odprtem svetu.

Slika 5.20: Mspf performančnost ABT drevesa v odprtem svetu.

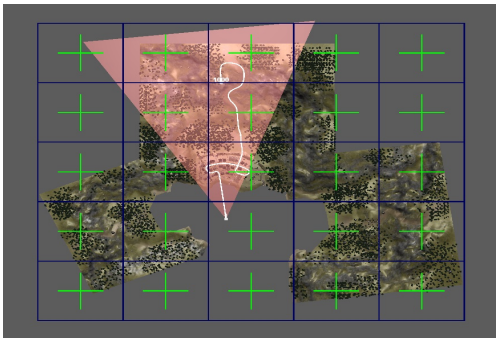
5.3.4 PVS

5.3.4.1 Odstranjevanje objektov

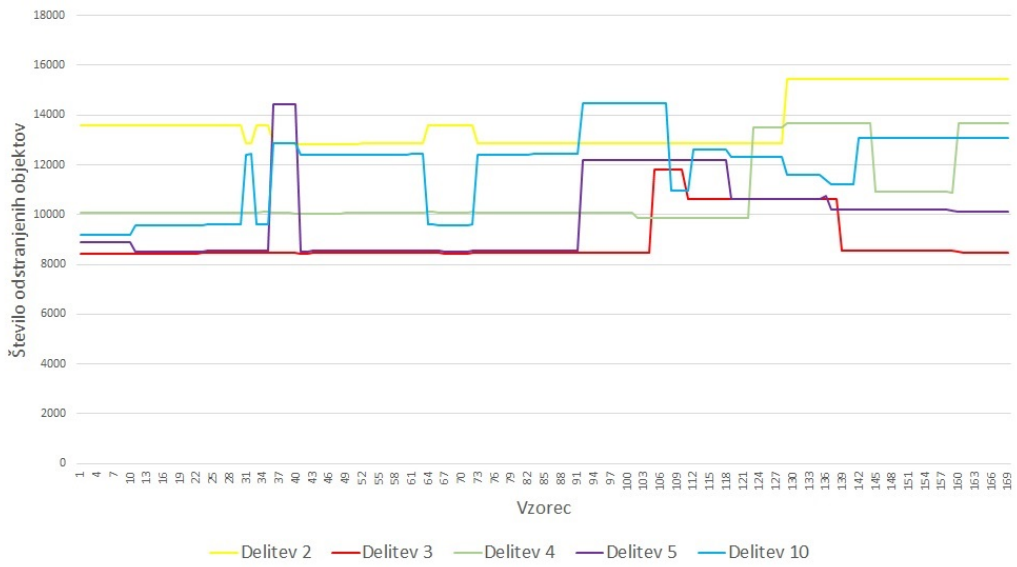
Količina odstranjevanja objektov PVS grafa na odprtem svetu sledi v primerjavi z ostalimi grafi scene povsem drugačnemu vzorcu (slika 5.23). Po odstranitvah sodeč je 2-kratna delitev boljša od vseh ostalih. Sledi ji 10-kratna delitev, medtem ko je 4-kratna delitev boljša od 3- in 5-kratne delitve. Med delitvami je opaziti rahlo podobnost, in sicer v okolici vzorca 37 ter pri intervalu 90-130, kjer se odstranjevanje za kratek čas poveča in nato pade. Izjema je 2-kratna delitev, kjer padca ni. Za vse delitve velja, da se pri vzorcu 37 kamera nahaja znotraj celice oziroma celic, ki obsegajo vas. Hiše in podobni objekti zakrivajo velik del upodobljenega vzorca, zato je temu primerno večje število odstranitev. Na intervalu med 90-130 kamera potuje izven vasi, zaradi česar vas zakriva večji del sveta. Primer delitve sveta s PVS grafom s 3- in 5-kratno delitvijo je upodobljen na slikah 5.21 in 5.22, kjer zeleni križci predstavljajo lokacijo kamere pri gradnji PVS grafa, celice pa so označene s temno modro barvo.



Slika 5.21: PVS graf globine 3 na odprtem svetu.



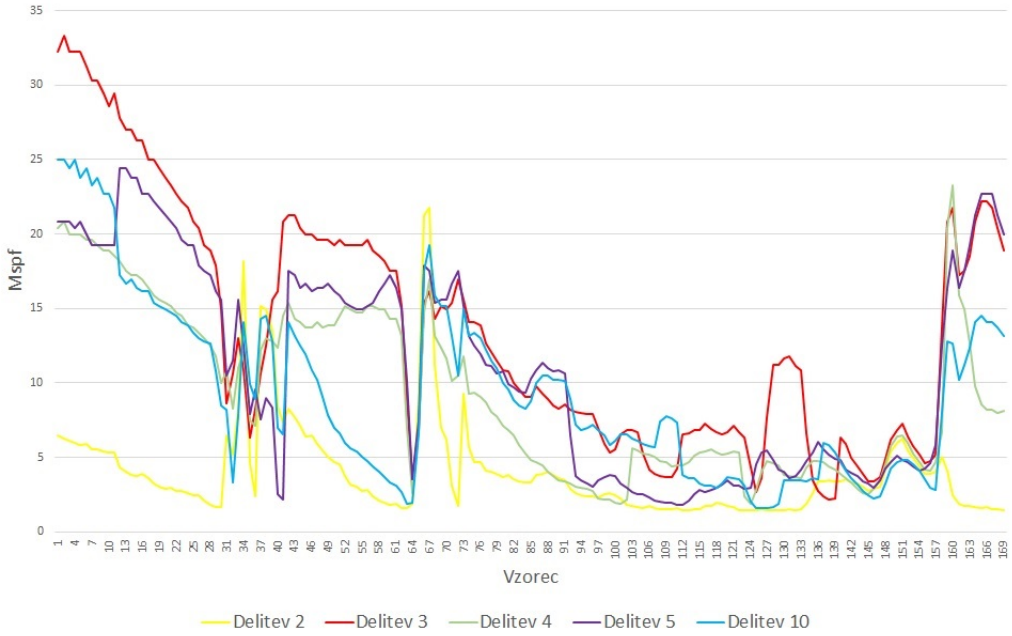
Slika 5.22: PVS graf globine 5 na odprtem svetu.



Slika 5.23: Število odstranjenih objektov z PVS grafom v odprtem svetu.

5.3.4.2 Performančnost

Hitrost PVS delitev (slika 5.24) je zelo različna, vendar v nasprotju z odstranjevanjem objektov bolj sledi nekakšnemu vzorcu. Kljub temu da večje delitve zahtevajo več procesiranja na ravni grafa, se to ne odraža v isti meri pri hitrosti. 10-kratna delitev je na primer dosti boljša od 3-kratne delitve. Glede na to da je hitrost neposredno povezana s časom upodabljanja, lahko sklepamo, da 3-kratna delitev upodobi večje število objektov. Ampak za to delitev velja tudi, da je bolj konsistentna pri upodabljanju, medtem ko se pri ostalih delitvah pojavljajo napake (predvsem odsotnost upodabljanja objektov) (sliki 5.25 in 5.26). Ker je pravilnost upodabljana bolj pomembna od hitrosti, je kljub slabšim rezultatom 3-kratna delitev edina uporabna možnost.



Slika 5.24: Mspf performančnost PVS grafa v odprtem svetu.

Slika 5.25: Nepravilnost upodabljanja pri PVS grafu delitve 5 v odprtem svetu (glej slika 5.26).

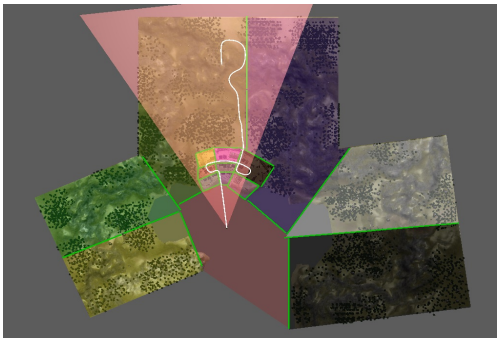
Slika 5.26: Pravilnost upodabljanja pri PVS grafu delitve 3 v odprtem svetu.

5.3.5 Portali

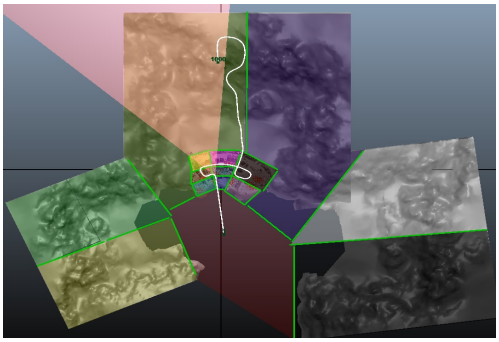
5.3.5.1 Odstranjevanje objektov

Količina odstranjevanja je predvsem odvisna od števila vidnih con, ki ustrezajo terenu. Celotno število con v svetu je 15. Na začetku scenarija je kamera obrnjena proti središču sveta, s čimer se odstrani stranski teren, ne pa vasi in terena v ozadju (slika 5.27). Odstranjevanje narašča do vzorca 30 (slika 5.28), kjer igralec vidi samo levo polovico terena za vasjo. Temu sledi nenaden padec pri vzorcu 32 (slika 5.29), kjer se vidnemu polju doda še levi teren. Povečanje odstranitev se ponovno poveča pri vzorcu 38 (slika 5.30), ko se kamera obrne nazaj proti terenu za vasjo. Z dodajanjem vedno večjega kosa celotnega terena odstranitve od tega vzorca dalje padajo in dosežejo minimum pri vzorcu 42 (slika 5.31).

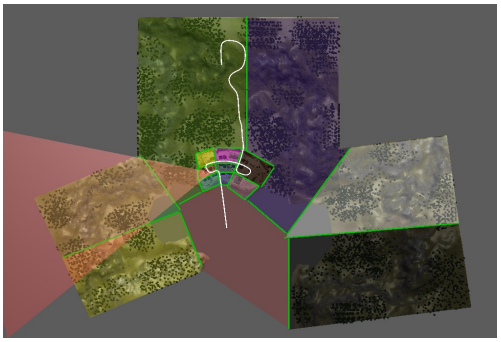
Kamera nato naredi zavoj, s čimer se poveča število odstranitev do vzorca 72. Sledi strm padec pri vzorcu 76 (slika 5.32), kjer je viden celoten teren za vasjo. Pri interval med vzorcema 115 in 123 je zabeležen vzpon in padec, saj se vidnemu polju doda desna polovica terena, ki se nato hitro odstrani. Globalni maksimum je zabeležen pri vzorcu 156 (slika 5.33), kjer je kamera obrnjena stran od središča sveta. Najmanjša odstranitev je na koncu scenarija pri vzorcu 162 (slika 5.34),



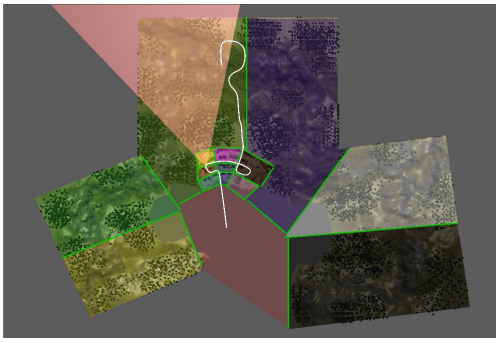
Slika 5.27: Vzorec 1 pri portalih v odprtem svetu.



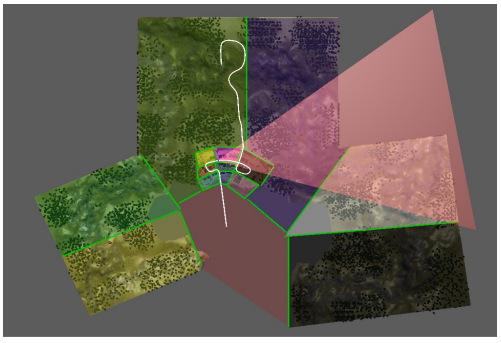
Slika 5.28: Vzorec 30 pri portalih v odprtem svetu.



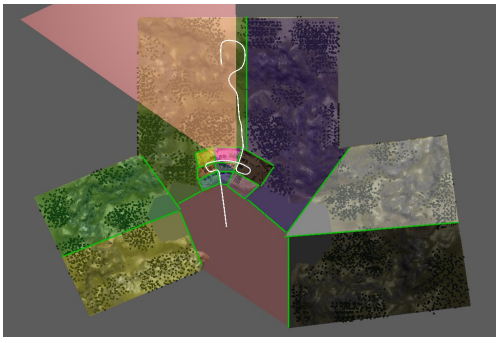
Slika 5.29: Vzorec 32 pri portalih v odprtem svetu.



Slika 5.30: Vzorec 38 pri portalih v odprtem svetu.



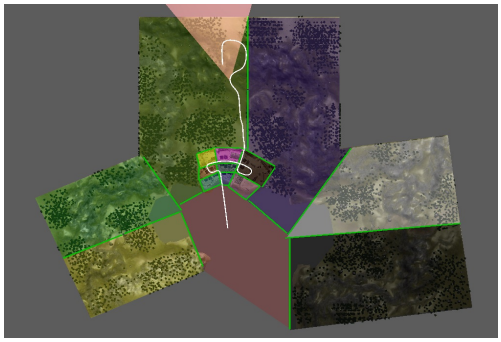
Slika 5.31: Vzorec 42 pri portalih v odprtem svetu.



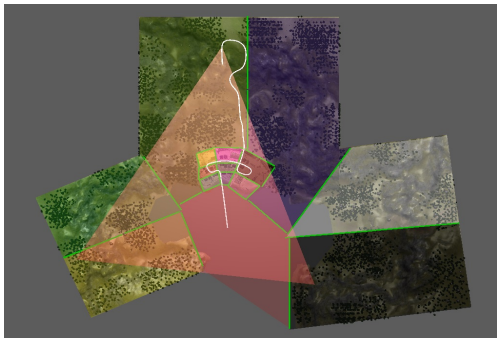
Slika 5.32: Vzorec 76 pri portalih v odprtem svetu.

kjer kamera gleda na celotno vas in na nivoju grafa ni praktično zabeležena nobena

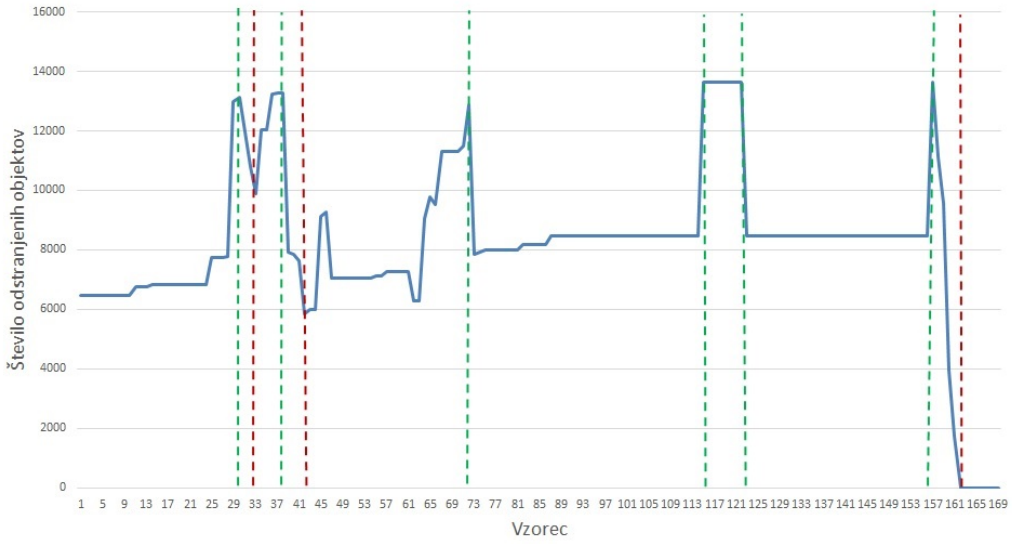
odstranitev. Spreminjanje odstranjevanja je razvidno na sliki 5.35.



Slika 5.33: Vzorec 156 pri portalih v odprtem svetu.



Slika 5.34: Vzorec 162 pri portalih v odprtem svetu.

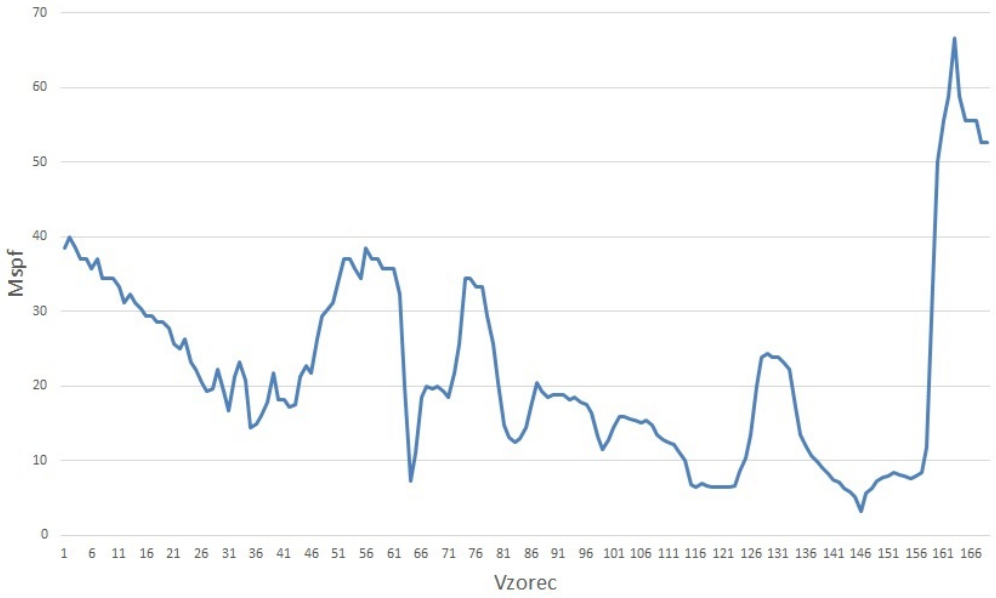


Slika 5.35: Število odstranjenih objektov s portali v odprtem svetu.

5.3.5.2 Performančnost

Maksimumi v performančnosti (slika 5.36) so na splošno povezani s količino odstranjenih objektov, vendar ta povezava ni tako očitna kot pri linearnemu seznamu. Z uporabo postavitve, ki bi natančno ustrezala scenariju, bi seveda lahko zelo izboljšali končni rezultat. Žal pa je postavljanje portalov v odprtem svetu zelo

težavno, saj ne obstajajo majhni in omejeni prehodi med področji. Poleg tega bi drugačna postavitev portalov preveč omejevala pot kamere v odprtem svetu in bi razveljavila scenarij testiranja.



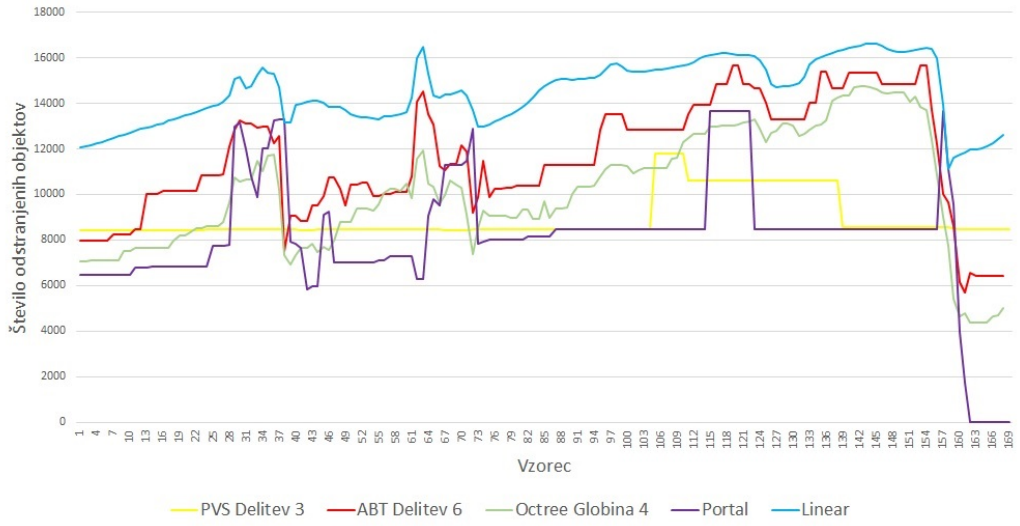
Slika 5.36: Mspf performačnost portalov v odprtem svetu.

5.3.6 Primerjava

5.3.6.1 Odstranjevanje objektov

Na sliki 5.37 so glede odstranjevanja objektov predstavljene najboljše različice vsake izmed testiranih oblik grafov. Najboljše rezultate nudi linearen seznam, saj je odstranjevanje na nivoju kamere povsem natančno, medtem ko je sistem portalov najslabši. Portali se zaradi pomanjkanja majhnih koridorov, ki povezujejo področja, zelo težko prilagajo odprtim tipom sveta in so tudi edini graf scene, ki ne odstrani nobenega objekta pri vzorcu 162. Tekom celotnega scenarija ABT premaga Octree globine 4 za okoli 2000 objektov. Do tega pride predvsem zato, ker se ABT področja lažje prilagajajo dejanski geometriji. PVS graf se z uporabo prekrivanja dobro odreže, čeprav slabše v primerjavi z bolj kompleksnimi rešitvami kot je na primer Octree. Tekom izvajanja scenarija je vidno polje velikokrat usmerjeno

na takšen način, da je velik del sveta zakrit, recimo znotraj vasi.

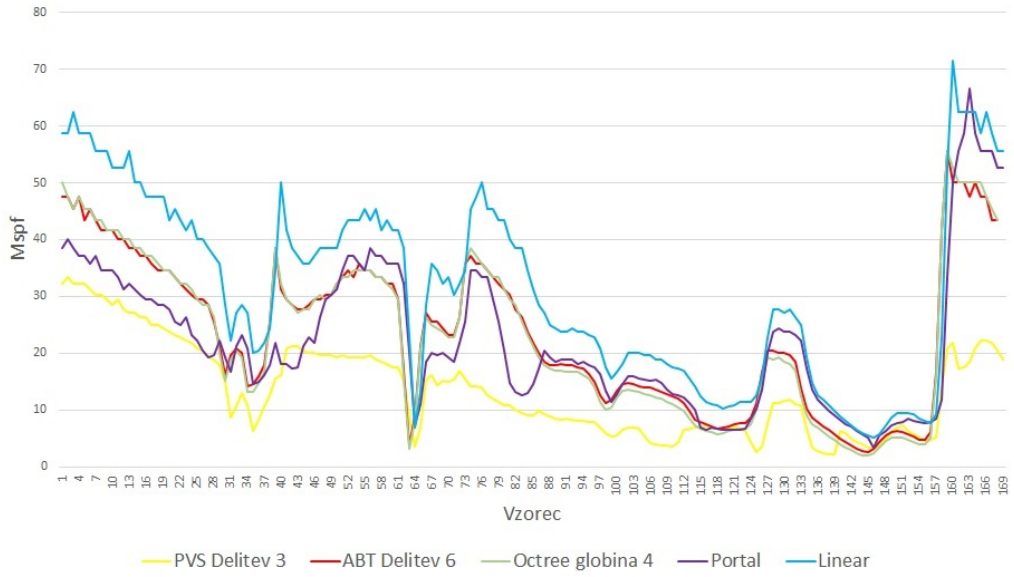


Slika 5.37: Število odstranjenih objektov najboljših predstavnikov v odprtem svetu.

5.3.6.2 Performančnost

Po performančnosti sodeč je daleč najboljši PVS graf (slika 5.38), saj konsistentno premaga vse grafe scen, in sicer v povprečju za več kot 20 milisekund. Na koncu izvajanja scenarija, ko je obremenitev največja, pa doseže celo razliko 30 milisekund. Razlog je predvsem v dobrem izkoriščanju prekrivanja objektov, s čimer se na koncu scenarija odstrani dobršen del sveta. Na drugem mestu je sistem portalov, ki kljub najslabšemu odstranjevanju objektov na splošno premaga obe binarne razdelitve sveta. Podobna ugotovitev velja tudi za graf PVS. Pri temu se odpira vprašanje, ali je odstranjevanje objektov v odprtem tipu sveta res primarnega pomena. ABT in Octree sta večino časa scenarija enako hitra, razen pri vzorcih večje obremenitve (na primer začetek ali konec scenarija), kjer ima ABT majhno prednost. Kljub vsem navedenim ugotovitvam ni jasna odločitev o temu, kateri graf naj bi bil najboljši za odprti svet. PVS in sistem portalov res dosegata veliko boljše rezultate, vendar so rezultati zelo odvisni od poteka scenarija oziroma igralca. Majhne spremembe poti bi lahko hitro uvedle napake v upodabljanju, predvsem pri grafu PVS in nekoliko manj pri portalih. Napake bi se predvsem

odražale v manjkajočih oziroma ne-upodobljenih delih sveta.



Slika 5.38: Mspf performančnost najboljših predstavnikov v odprtem svetu.

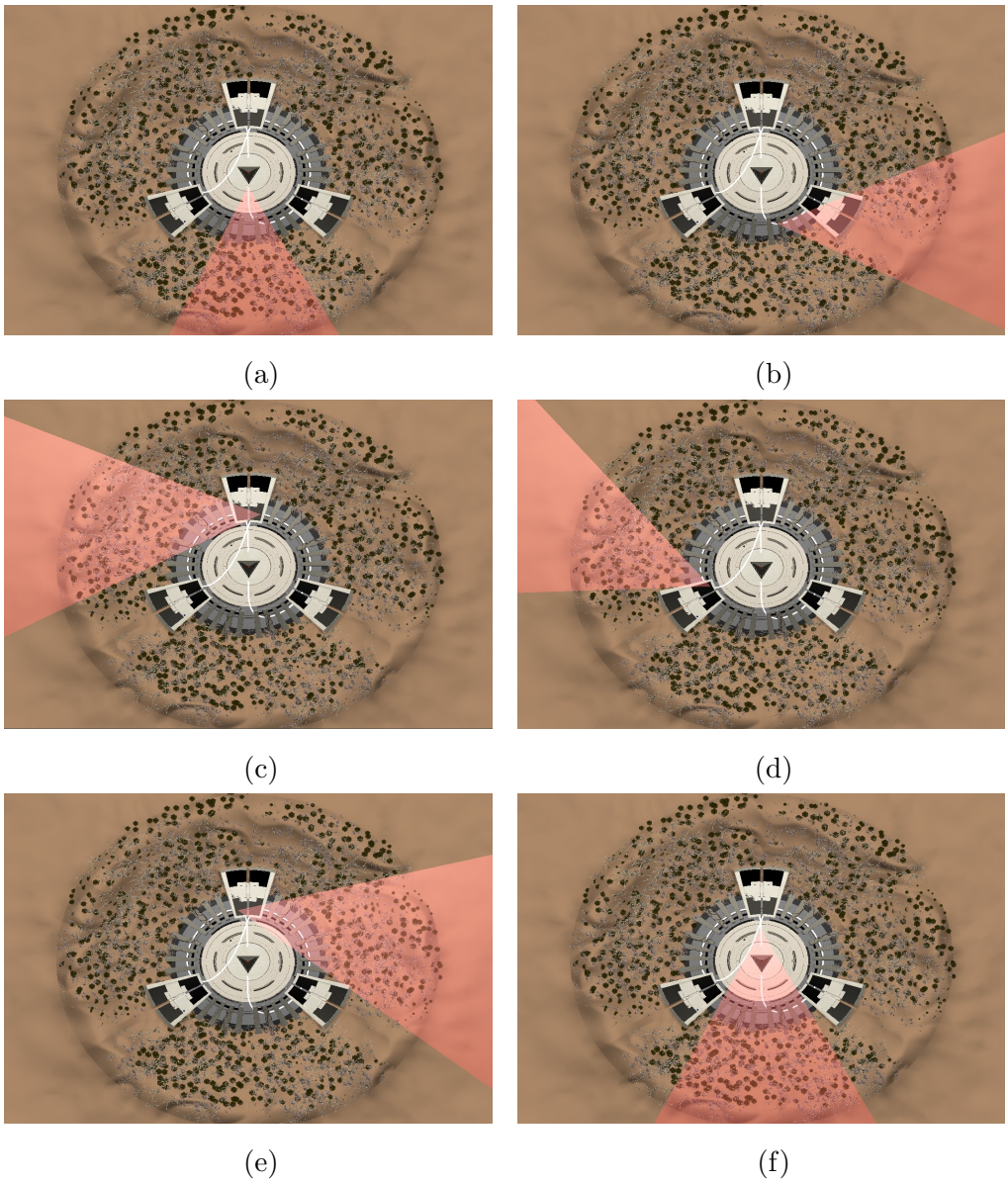
5.4 Pol-odprti svet

5.4.1 Linearen seznam

5.4.1.1 Odstranjevanje objektov

Linearen seznam v tej obliki sveta ne izkorišča številne priložnosti za odstranjevanje, ki se pojavijo zaradi same narave pol-odprtega sveta. V svetu je par odsekov pri vzorcih 0, 15, 37, 58, 75 in 80 (slika 5.39) s pogledom na odprti svet v zunanjem obroču, kjer se število odstranjenih objektov zelo poslabša. Celotno odstranjevanje se izvaja na nivoju kamere, kjer so izmed vseh grafov vrednosti največje. Minimumi v številu odstranitev pri vzorcih ustrezajo trenutkom, ko se kamera nahaja na zunanjem obroču s pogledom na zunanji svet.

Maksimumi v številu odstranitev se pojavijo pri vzorcih 8, 28, 39, 54, 65 in 77 (slika 5.40), ko kamera umakne pogled iz zunanjega sveta proti stolpom. V teh trenutkih vidno polje kamere ne vsebuje objektov, ki so posejani na terenu, kot na primer drevesa, kamenje in hlodi. Število odstranitev tekom scenarija je prikazano na sliki 5.41.

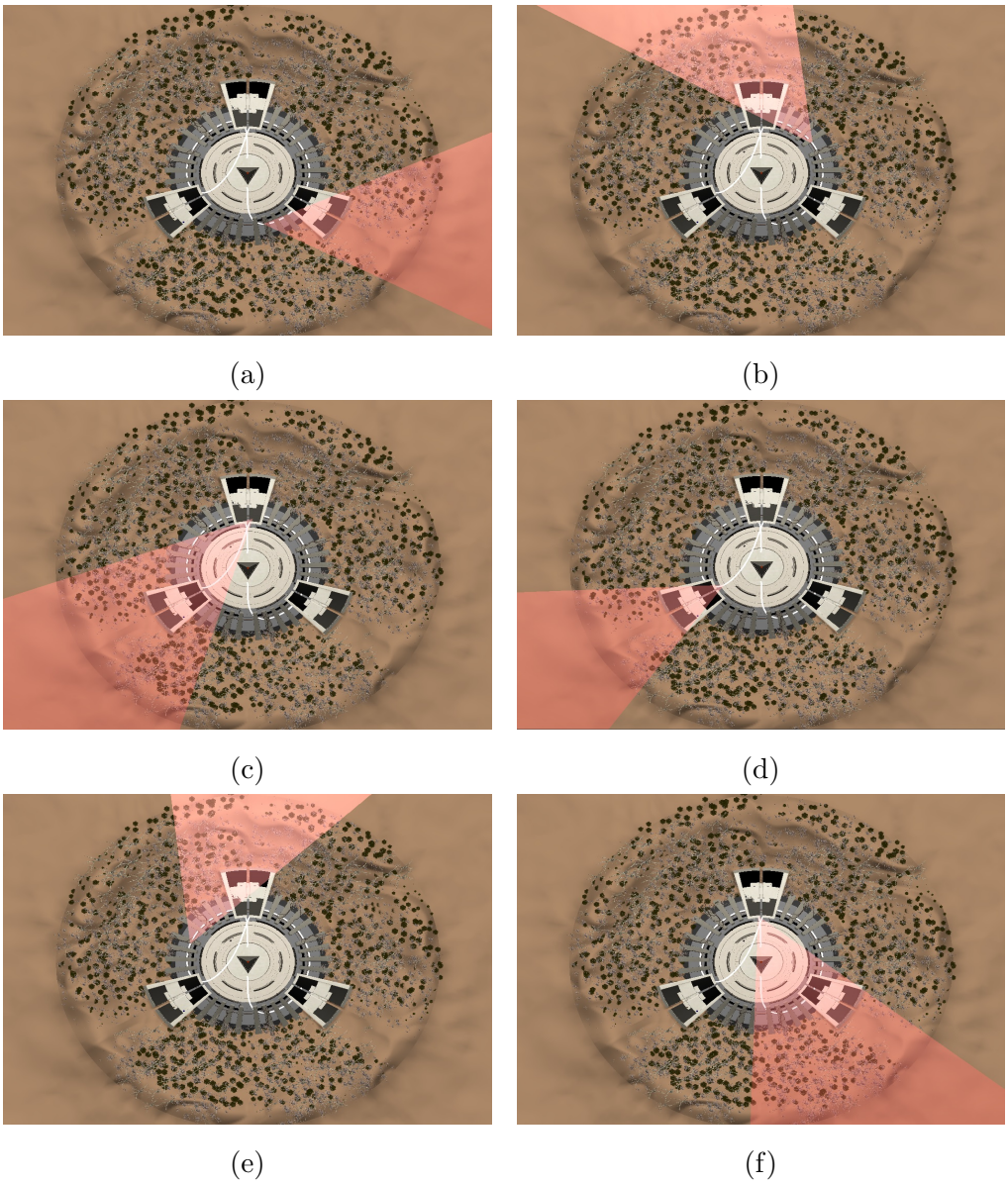


(a) (b)

(c) (d)

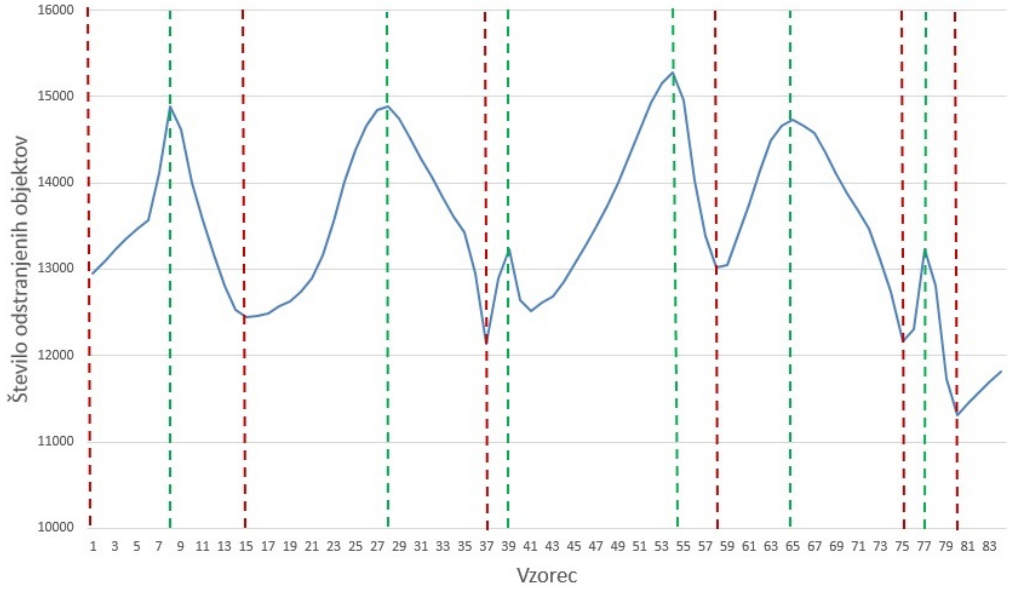
(e) (f)

Slika 5.39: Minimumi števila odstranjenih objektov pri linearnem seznamu v pol-odprtem svetu.



(a) (b) (c) (d) (e) (f)

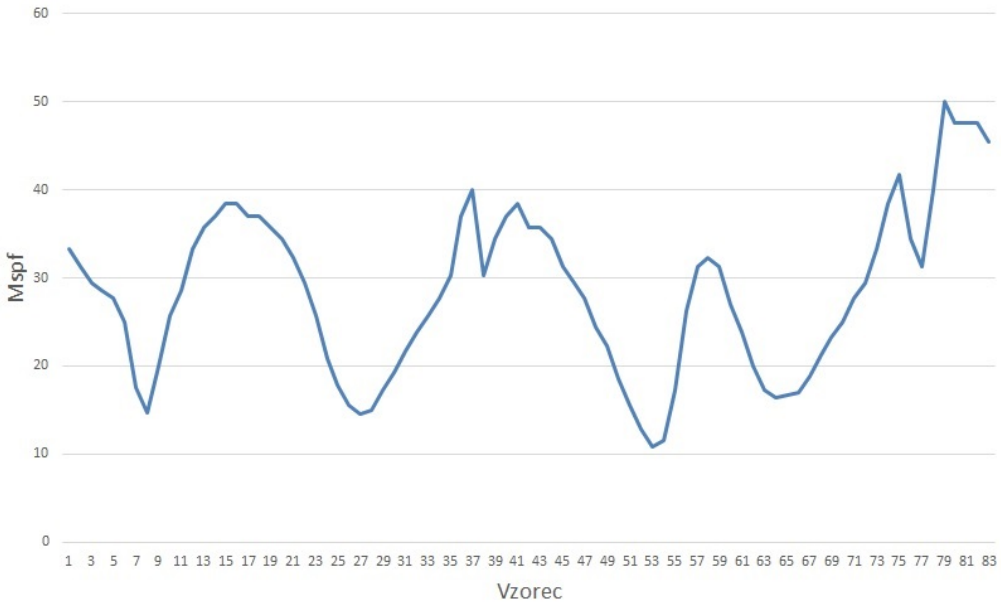
Slika 5.40: Maksimumi števila odstranjenih objektov pri linearnem seznamu v pol-odprtem svetu.



Slika 5.41: Število odstranjenih objektov z linearnim seznamom v pol-odprtem svetu.

5.4.1.2 Performančnost

Hitrost linearnega seznama je ponovno linearno odvisna od števila odstranitev (slika 5.42). Edina možnost za povečanje performančnosti je zmanjšanje števila objektov na točkah v scenariju, kjer kamera opazuje zunanji svet. Na podlagi slik se lahko med drugim opazi vpliv širine vidnega polja. Z zmanjšanjem širine vidnega polja bi lahko zmanjšali obseg padcev v količini odstranjenih objektov in obratno.

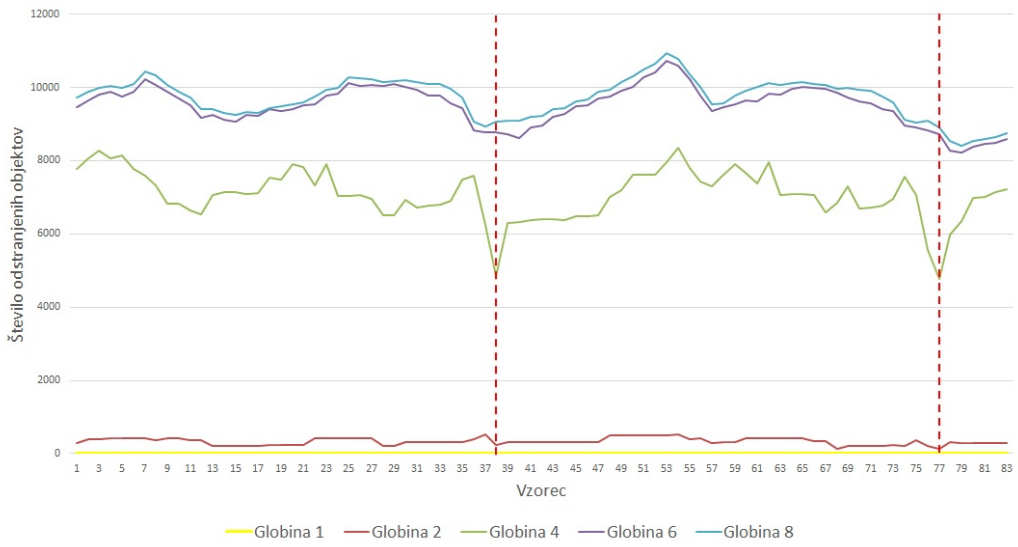


Slika 5.42: Mspf performančnost najboljših predstavnikov v odprtem svetu.

5.4.2 Octree

5.4.2.1 Odstranjevanje objektov

Uporabljen scenarij pri testiranju pol-odprtega sveta vodi kamero po krožnici blizu centra sveta. Kar pomeni, da je pri dobršnem delu poti kamera postavljena izven centra sveta s pogledom stran od središča. Takšen potek in narava sveta omogočata veliko odstranjevanj na nivoju Octree drevesa (slika 5.43). Tudi v tem primeru se odstranjevanje na nivoju grafa ustali pri globini 8, medtem ko predstavlja globina 4 rezultate med dvema ekstremoma, globino 8 in globino 1.



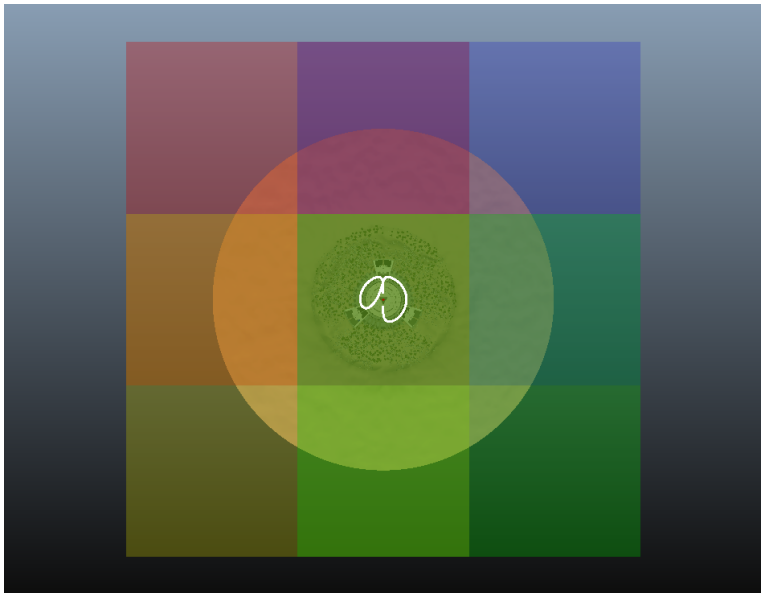
Slika 5.43: Število odstranjenih objektov z Octree drevesom v pol-odprtem svetu.

Zaradi same narave ohlapnega Octree drevesa globina 1 ne dopušča možnosti za kakršno koli odstranjevanje. Na sliki 5.44 so prikazani oktanti globine 1, kjer je vsak oktant označen z drugo barvo in je pot kamere obarvana z belo barvo. Med izvajanjem scenarija se kamera ves čas nahaja na presečišču oktantov (slika 5.44), zaradi česar je vseh devet oktantov globine 1 (slika 5.47) vidnih. V temu primeru povečevanje obsega oktantov za faktor 1,5 v celoti onemogoča odstranjevanje na nivoju grafa. Če bi se kamera pomikala veliko bolj izven središča sveta, bi bilo odstranjevanje možno tudi pri tej globini.

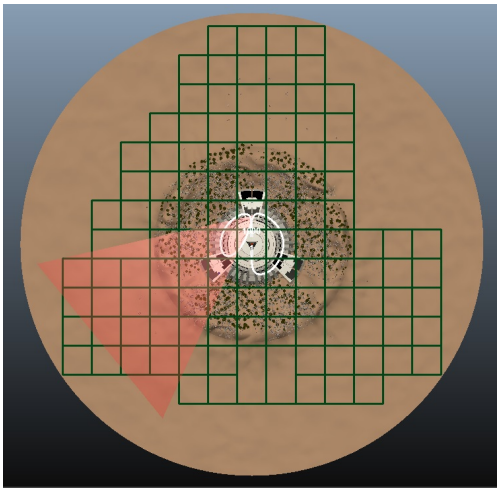
Odstranjevanje se končno ustali pri globini 8. To pa zato, ker celic na globini 8 ni več mogoče zapolniti v dovolj velikem številu, da bi to lahko vplivalo na odstranjevanje. Vprašanje, ali so oktanti ustvarjeni, je odvisno od velikosti oktantov in položaja objektov.

Čeprav je tekom scenarija več minimumov (slika 5.43) in ustrezajo podobnim pogojem, kot veljajo za linearen seznam, in sicer je to položaj na zunanjem obroču in usmerjenost kamere proti terenu, sta najbolj opazna minimuma pri vzorcih 38 in 77 (sliki 5.45 in 5.46). Na teh dveh položajih, ko se pomika iz zunanjega obroča proti središču sveta, kamera vidi največje število celic.

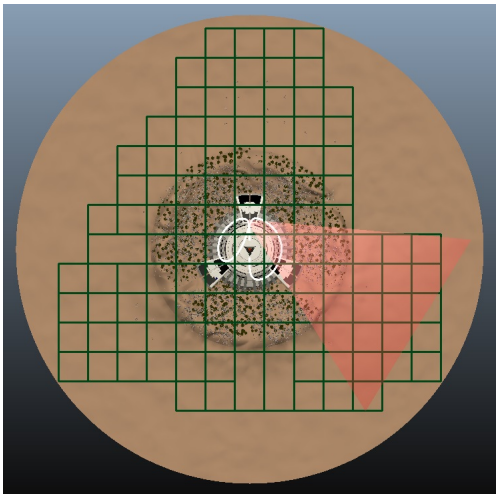
Maksimumi v številu vidnih oktantov (slika 5.47) so neposredno povezani z



Slika 5.44: Oktanti ohlapnega Octree drevesa globine 1 v pol-odprtem svetu.

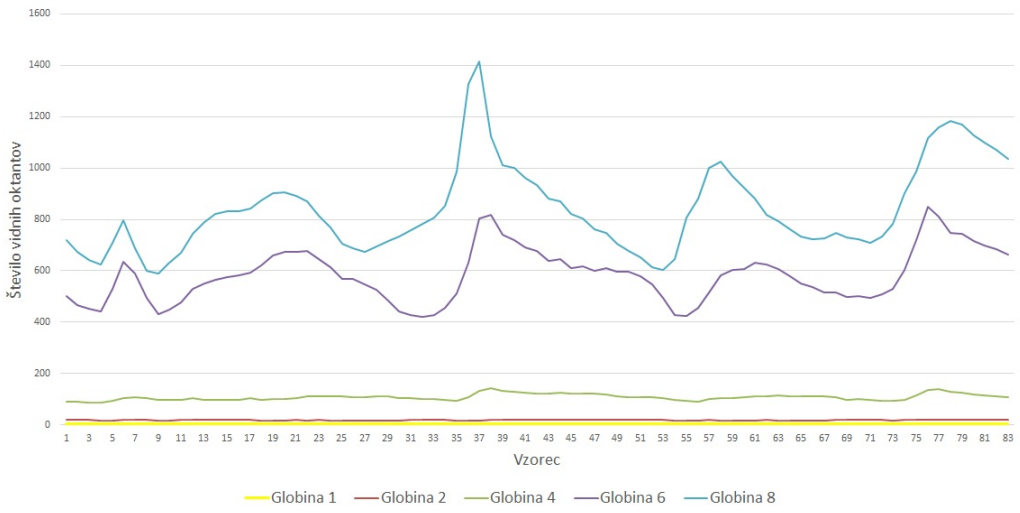


Slika 5.45: Vzorec 38, Octree, pol-odprti svet .



Slika 5.46: Vzorec 77, Octree, pol-odprti svet.

minimumi v številu odstranitev. Najbolj se opazi skok pri globinah 6 in 8. Ta trenutek ustreza pomikanju kamere proti središču.

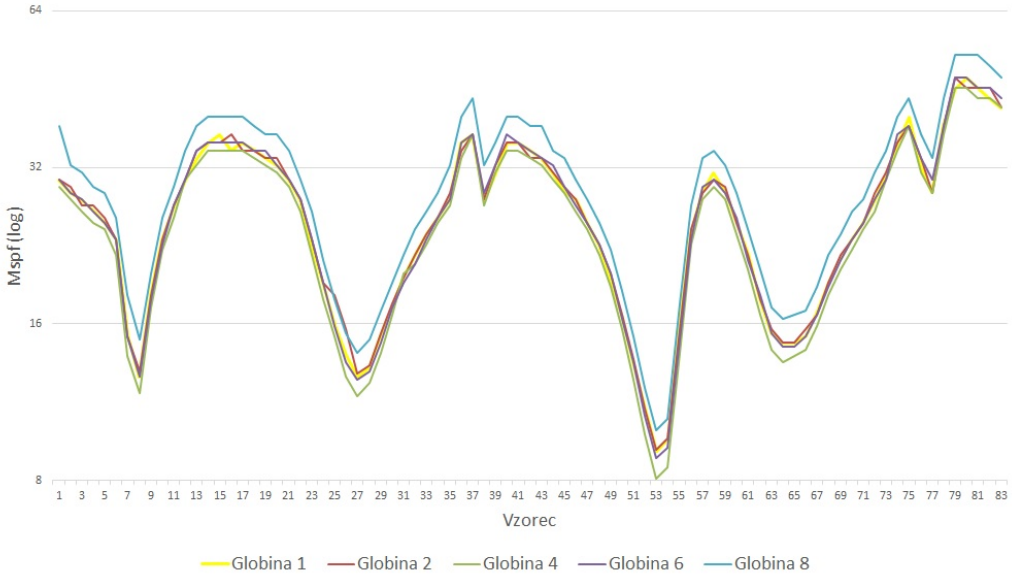


Slika 5.47: Število vidnih oktantov pri Octree drevesu v pol-odprtem svetu.

5.4.2.2 Performančnost

Tudi performančne karakteristike so skladne s količino odstranitev (slika 5.48). Opazi se, da ima veliko število celic zelo negativen vpliv na performančnost, medtem ko je dejansko število odstranjenih objektov majhno. Najbolj optimalni rezultati so pridobljeni z globino 4, ki najbolj efektivno izkoristi prihranke z odstranjevanjem na višjem nivoju in hkrati ne porabi preveč časa za odstranjevanje na nivoju kamere.

Tekom celotnega poteka poti kamere se Octree drevo globine 4 na splošno najbolje odreže. Z izjemo globine 8, so si hitrosti preostalih grafov pri precejšnemu delu poti kamere podobne. Sodeč po grafu na sliki 5.43 je največja prednost globine 4 dosežena pri majhnemu številu odstranjevanj na nivoju kamere, kar ustreza usmeritvi kamere stran od zunanjega sveta.

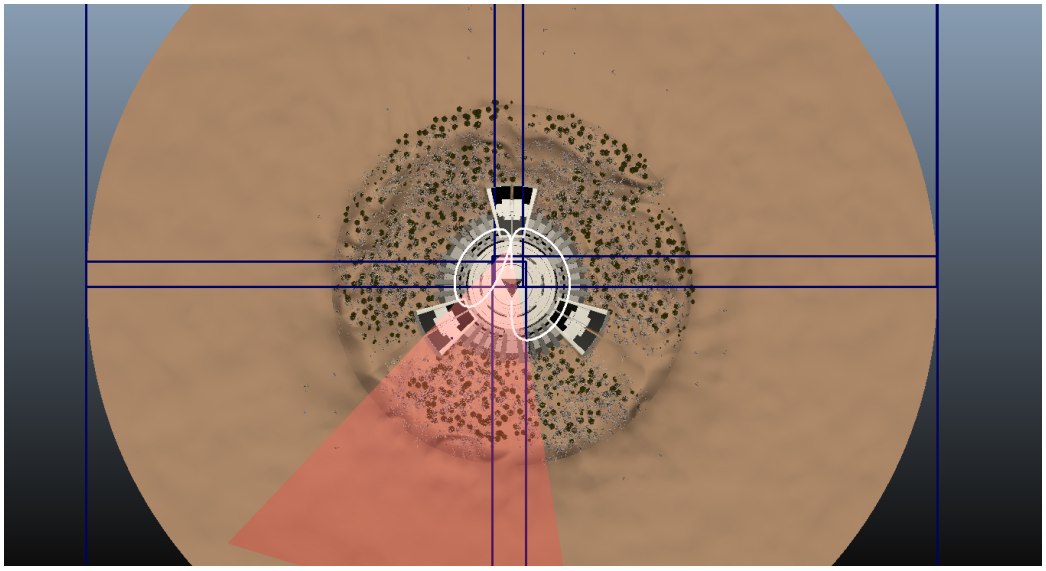


Slika 5.48: Mspf performačnost Octree drevesa v pol-odprtem svetu.

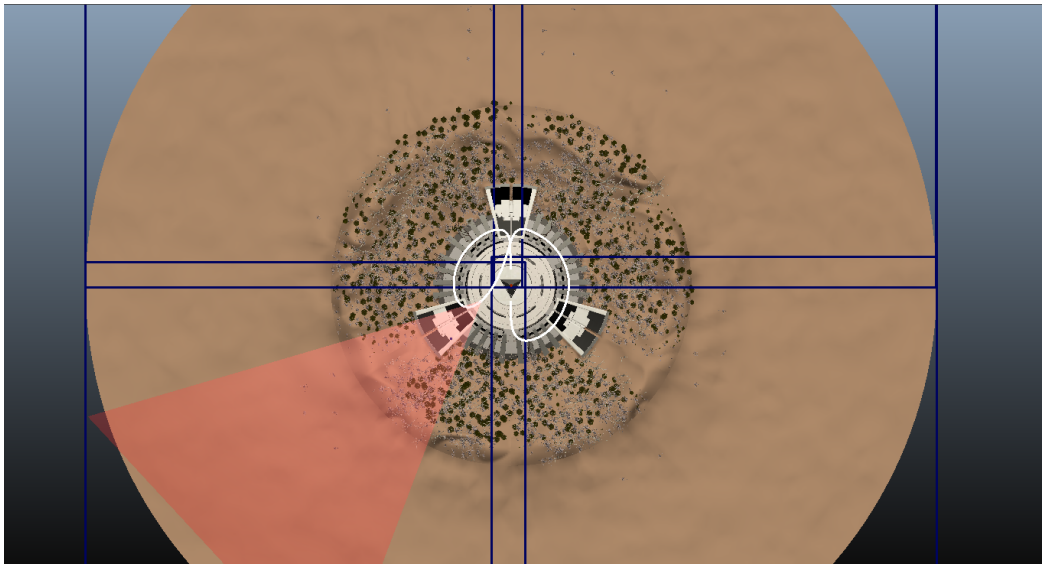
5.4.3 ABT

5.4.3.1 Odstranjevanje objektov

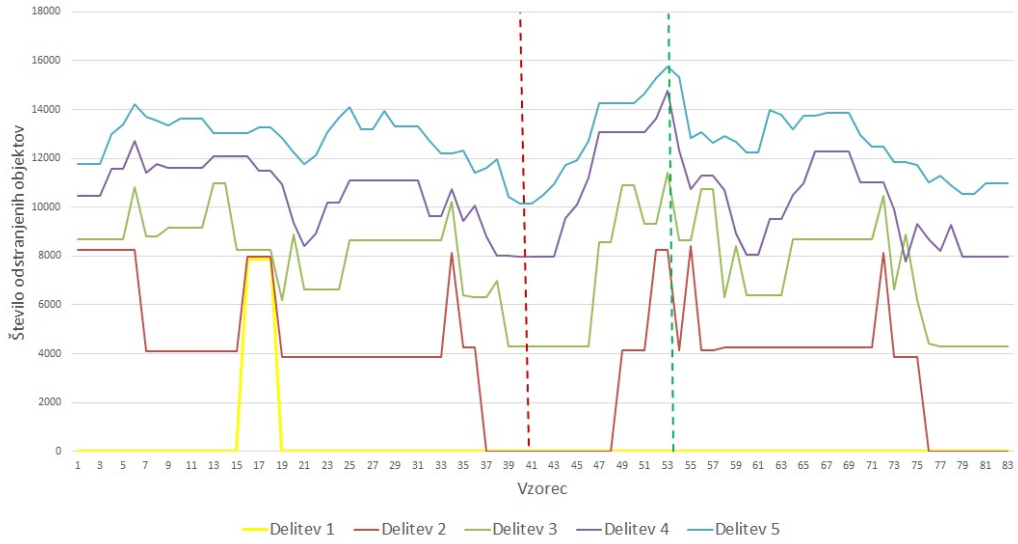
Pri vseh delitvah tudi ABT odstranjuje manjši del objektov, ko je kamera na zunanjih lokih zgradbe in obrnjena proti zunanjosti (slika 5.51). Prav tako je število odstranitev obratno sorazmerno s številom vidnih področij (slika 5.52). Razlika v količini odstranitev med delitvama 4 in 5 pomeni, da je kljub velikemu številu novo dodanih delitev ABT drevo uspelo zapolniti nova področja z zadostno količino objektov. Odstranjevanje se ne ustali niti pri globini 5 in bi se ga lahko še povečevalo; takšna globina bi bila kvečjemu uporabljena zaradi konsistence z ostalimi meritvami. Pri tem je treba poudariti, da je večje število odstranjenih objektov lahko tudi posledica novo nastalih objektov, ki nastanejo pri delitvah področij zaradi rezanja originalnih objektov. Vsem delitvam je skupen minimum pri vzorcu 41 (slika 5.49), ko kamera zavije proti središču sveta, in maksimum pri vzorcu 53 (slika 5.50), ko kamera zavije na stranski lok.



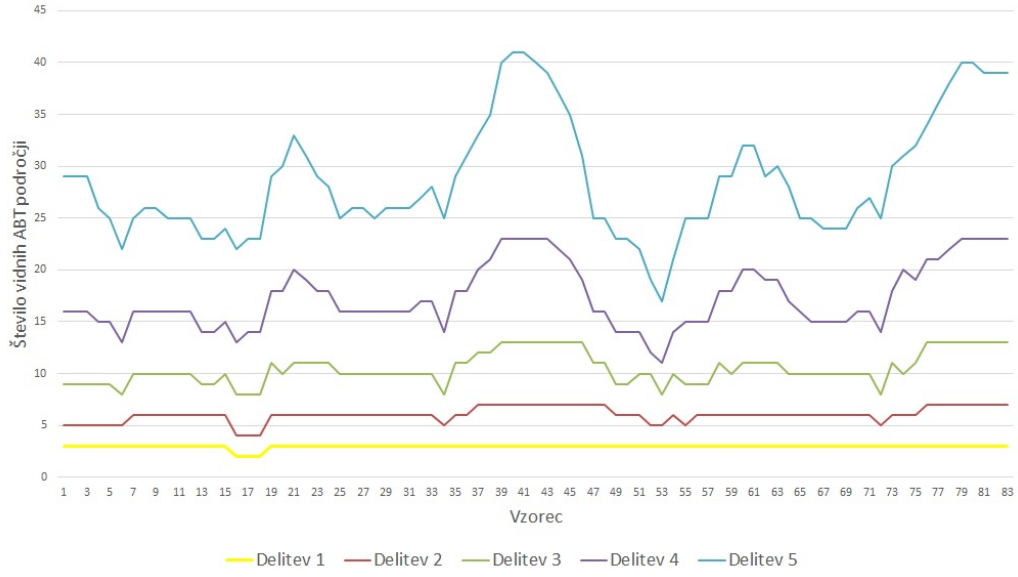
Slika 5.49: Vzorec 41 pri ABT drevesu z 2-kratno delitvijo v pol-odprtem svetu.



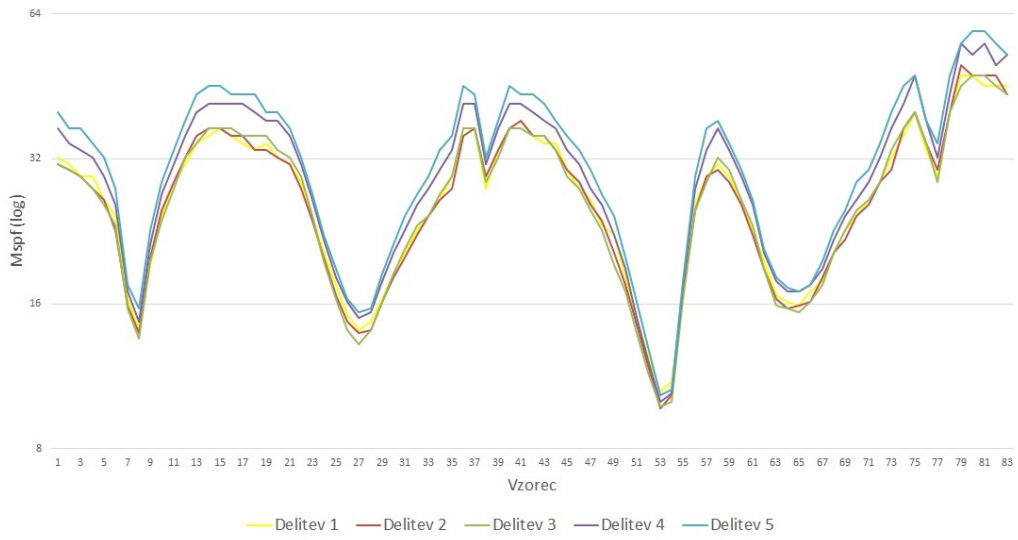
Slika 5.50: Vzorec 53 pri ABT drevesu z 2-kratno delitvijo v pol-odprtem svetu.



Slika 5.51: Število odstranjenih objektov z ABT drevesom v pol-odprtem svetu.



Slika 5.52: Število vidnih ABT področij v pol-odprtem svetu.



Slika 5.53: Mspf performančnost ABT drevesa v pol-odprtem svetu.

5.4.3.2 Performančnost

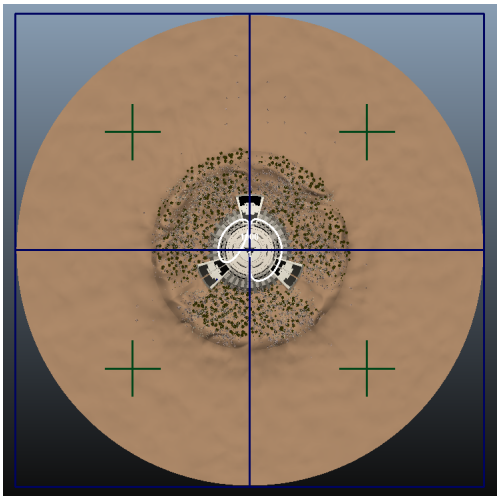
Kljub dobremu odstranjevanju z večanjem delitve ABT drevesa, to ne vpliva pozitivno na performančnost. Delitvi stopnje 4 in 5 sta lahko od 5 do 10 milisekund slabši v primerjavi z najbolj enostavno 2-kratno delitvijo. Razlog je v rezanju objektov, ki ga povzroči ABT drevo. Zaradi velike koncentracije objektov na terenu, ki je viden na lokih zgradbe, je rezanje objektov bolj opazno pri tej obliki sveta. Najboljši rezultati so pridobljeni s 3-kratno delitvijo, ko je kamera obrnjena stran od zunanjosti. Le-ti pa so kvečjemu za 1-2 milisekunde boljši od 2-kratne delitve, medtem ko je pri ostalih vzorcih na prvem mestu 2-kratna delitev (slika 5.53).

5.4.4 PVS

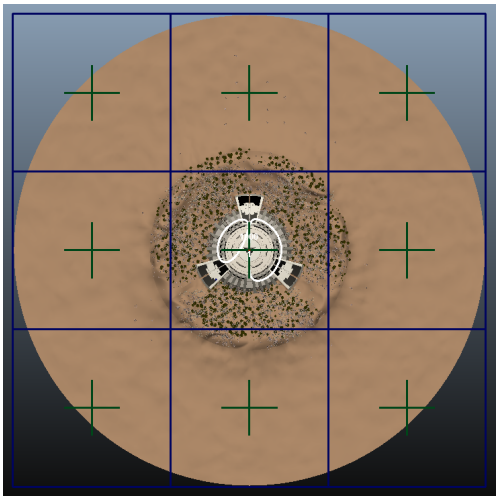
5.4.4.1 Odstranjevanje objektov

Rezultati odstranjevanja za vse delitve PVS grafa (slika 5.56) so si precej različni in ostajajo v veliki večini konstantni tekom izvajanja scenarija. Delitvi 3 in 5 sta praktično identični. Liho število delitev postavi celice na takšen način, da kamera ves čas izvajanja scenarija potuje znotraj ene same celice (slika 5.55). Vse delitve odstranjujejo večino objektov, ampak najboljši rezultati so zabeleženi pri 10-kratni

delitvi. Odstranjevanje prav tako ni odvisno od tega, ali se kamera nahaja znotraj lokov zgradbe ali ne. Visoke vrednosti odstranjevanja so možne zaradi velikega zakrivanja objektov. Če je kamera pri grajenju grafa postavljena na zunanjem terenu, je večina sveta zakritega z drevesi ali pa s samo zgradbo. Kamera, ki je postavljena znotraj objekta, je podvržena istim pogojem, saj je celotna zunanjost prav tako zakrita. Primer delitve s PVS grafom pri 2- in 3-kratni delitvi je prikazan na slikah 5.54 in 5.55, kjer zeleni križci predstavljajo lokacijo kamere pri gradnji PVS grafa, celice pa so označene s temno modro barvo.



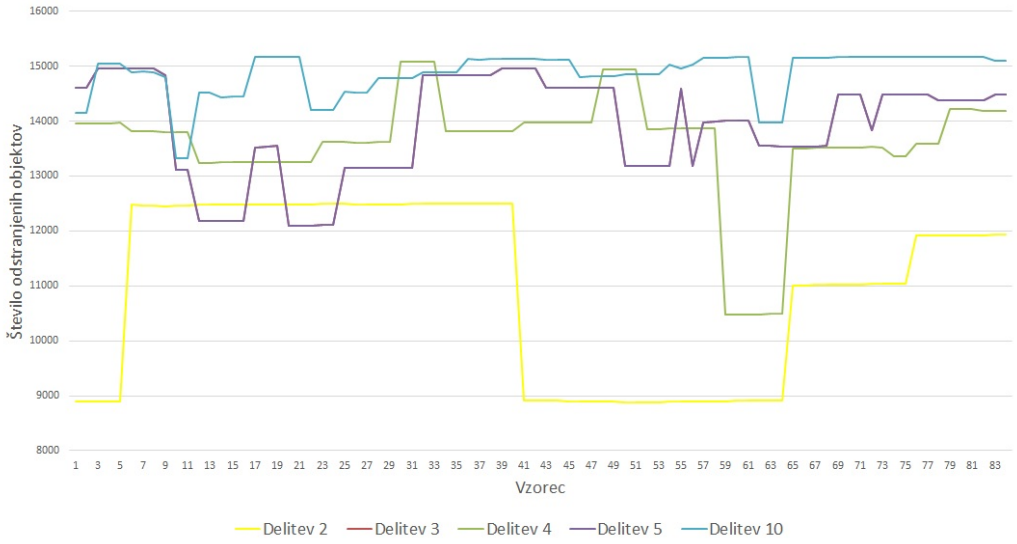
Slika 5.54: PVS graf 2-kratne delitve v odprtem svetu.



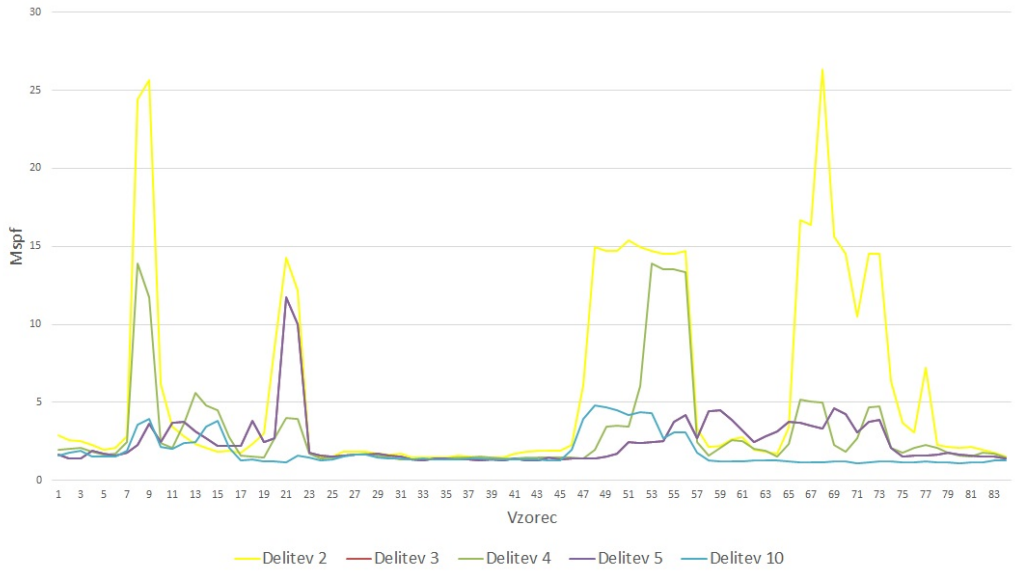
Slika 5.55: PVS graf 3-kratne delitve v odprtem svetu.

5.4.4.2 Performančnost

Hitrost je predvsem odvisna od števila odstranitev. Najslabše odstranjevanje doseže 2-kratna delitev, zaradi česar ima ta tudi najslabše performančne rezultate (slika 5.57). Najhitrejša je 10-kratna delitev, ki ima tudi najboljše odstranjevanje, medtem ko delitvi 3 in 5 dosegata skoraj identične rezultate. Konsistenca upodabljanja je najboljša pri 3- in 5-kratnih delitvah, kjer se pojavi nič ali zalo malo artefaktov. Primer pravilnega in nepravilnega uporabljanja 2- in 3-kratne delitve je prikazan na slikah 5.58 in 5.59.

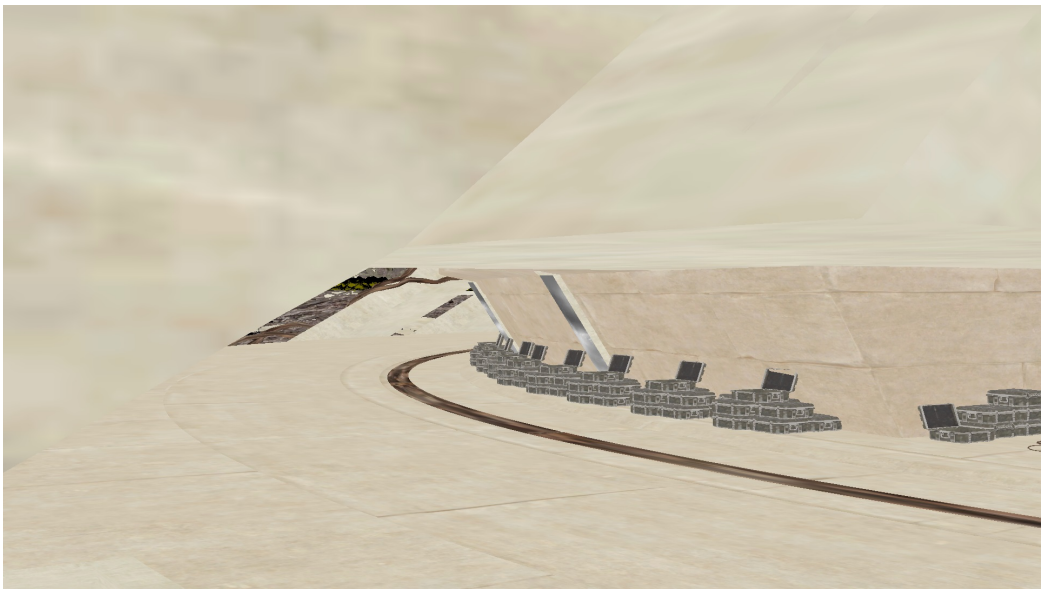


Slika 5.56: Število odstranjenih objektov s PVS grafom v pol-odprtem svetu.



Slika 5.57: Mspf performančnost PVS grafa v pol-odprtem svetu.

Slika 5.58: Nepravilnost upodabljana pri PVS grafu delitve 2 v pol-odprtem svetu (glej slika 5.59).

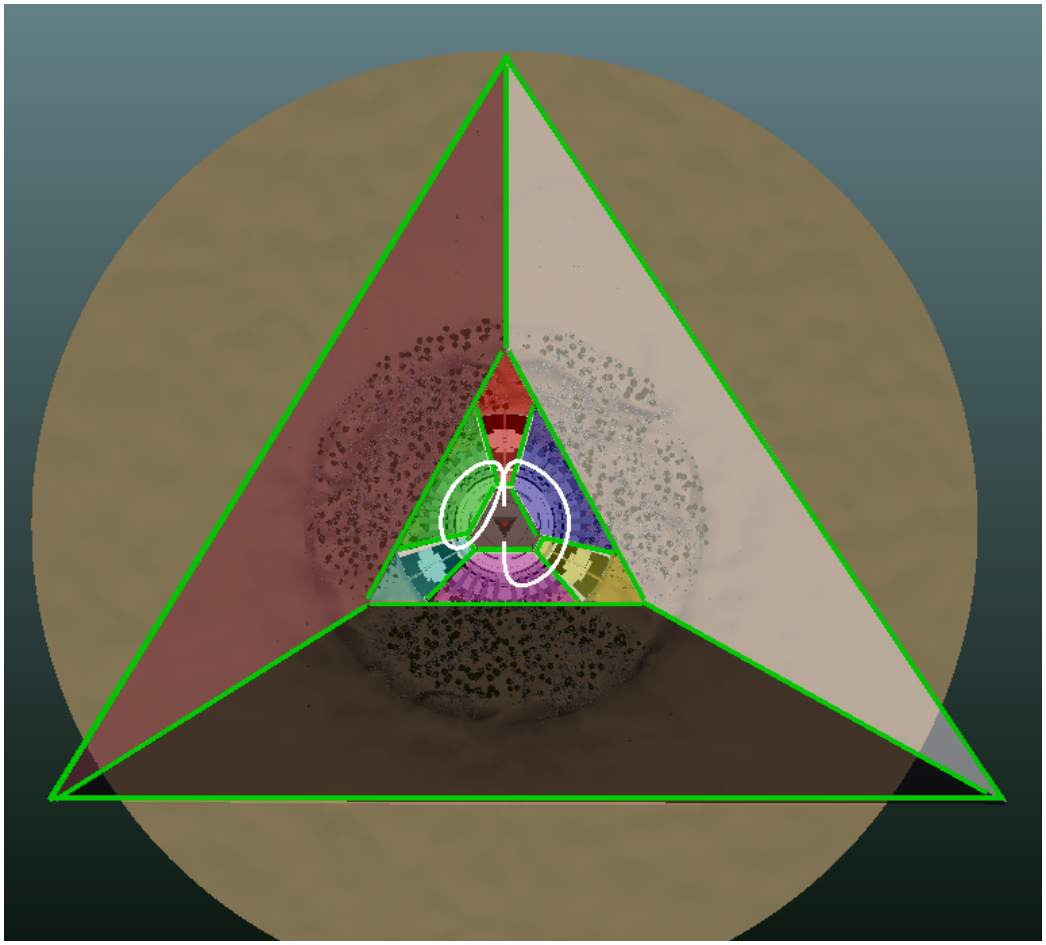


Slika 5.59: Pravilnost upodabljana pri PVS grafu delitve 3 v pol-odprtem svetu.

5.4.5 Portali

5.4.5.1 Odstranjevanje objektov

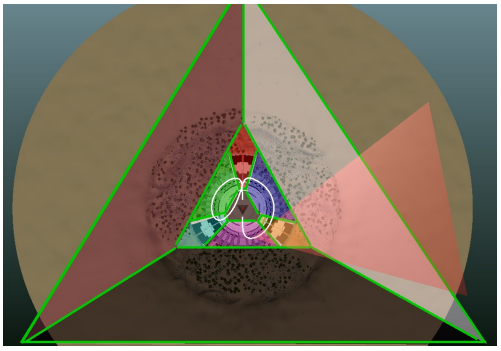
Portali so veliko bolje prilagojeni za pol-odprto obliko sveta, kjer se lahko zlahka omeji vidne dele prostora. Na sliki 5.60 je vsaka cona označena z drugo barvo, medtem ko so portali označeni s svetlo zelenimi črtami. Celotno število con je 10. Loki zgradbe so obdani s portali, ki loke povezujejo med seboj in s terenom.



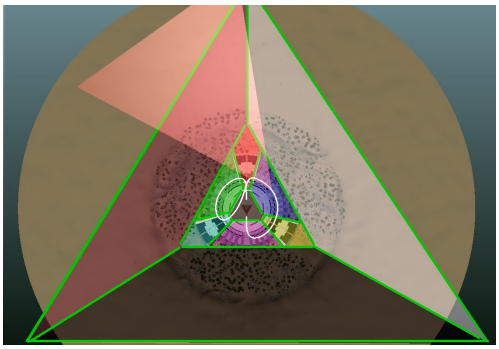
Slika 5.60: Cone in postavitev portalov v pol-odprtem svetu.

Odstranjevanje je večji del časa konstantno, vendar so pri vzorcih 13-15, 35-39 in 73-79 (slike 5.61, 5.62, 5.63) zabeleženi padci. Pri teh vzorcih kamera zajame največje število portalov in je s tem posledično odstranjevanje na nivoju grafa manjše. Pri vseh omenjenih intervalih kamera zajame dva zunanja portala, ki

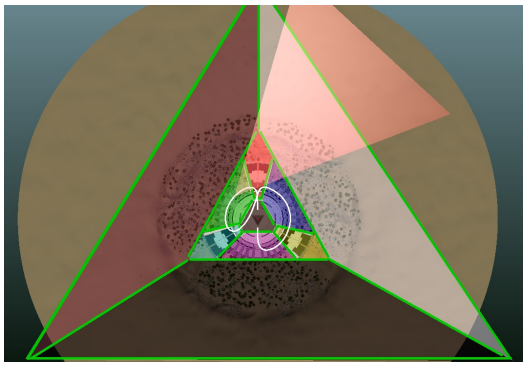
določata vidnost terena, in tri portale na zunanjem obroču zgradbe. Ti izločijo iz odstranjevanja dva loka zgradbe in stolp. S takšnim znanjem bi lahko postavili portale malo drugače, saj prehod iz enega loka zgradbe na drug preko stolpa ne bi smel upoštevati vidnosti terena na drugi strani. Odstranjevanje tekom scenarija je prikazano na sliki 5.64.



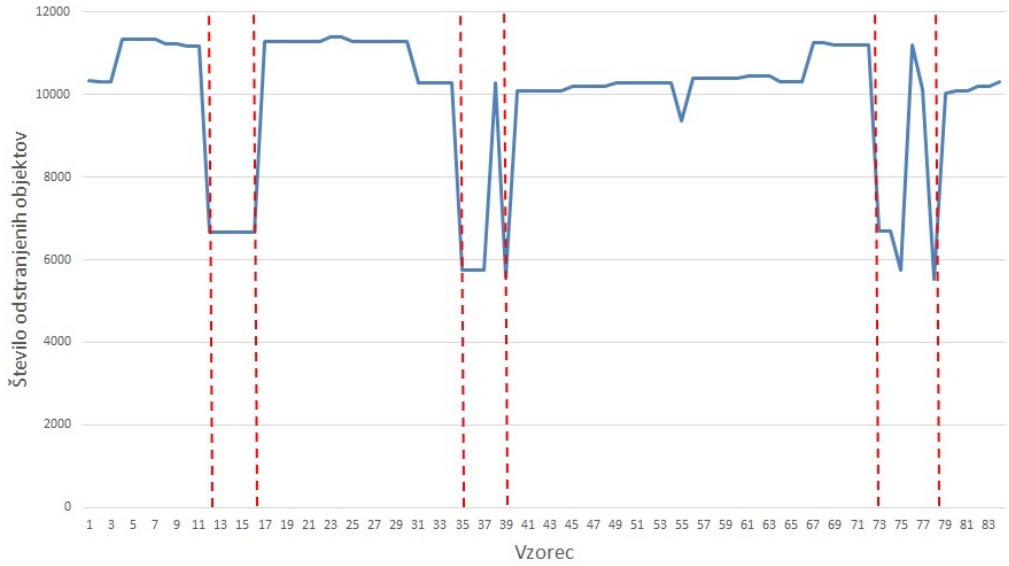
Slika 5.61: Vzorec 13 pri portalih v pol-odprtem svetu.



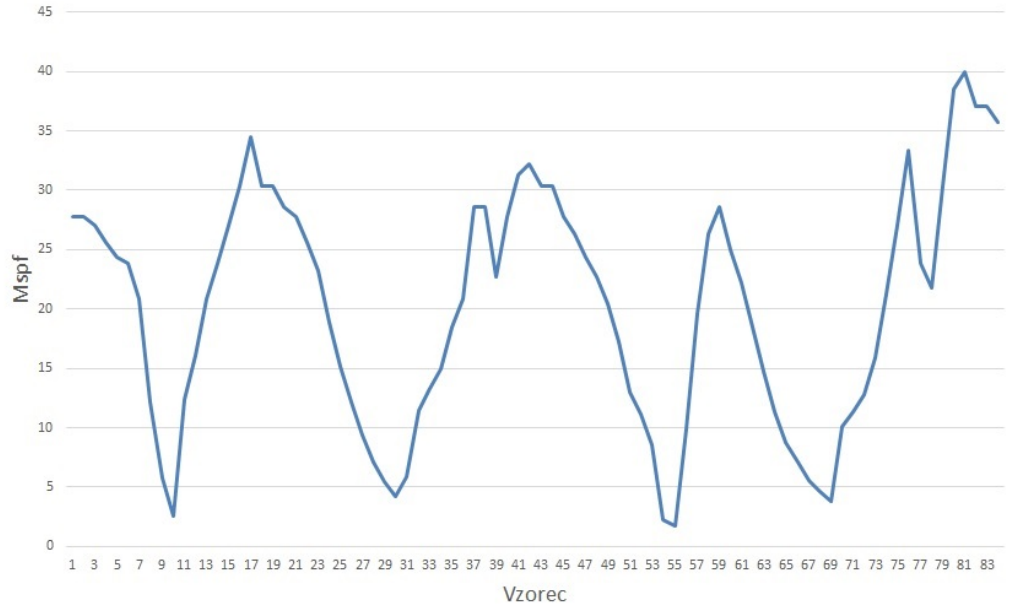
Slika 5.62: Vzorec 39 pri portalih v pol-odprtem svetu.



Slika 5.63: Vzorec 79 pri portalih v pol-odprtem svetu.



Slika 5.64: Število odstranjenih objektov s portali v pol-odprtem svetu.



Slika 5.65: Mspf performančnost portalov v pol-odprtem svetu.

5.4.5.2 Performančnost

Hitrost portalov (slika 5.65) je povezana s položajem kamere na zunanjem loku zgradbe in sledi trendu performančnosti ostalih grafov. Največji padci v hitrosti so povezani s številom vidnih portalov, ki ustrezajo zunanjosti (slike 5.61, 5.62, 5.63).

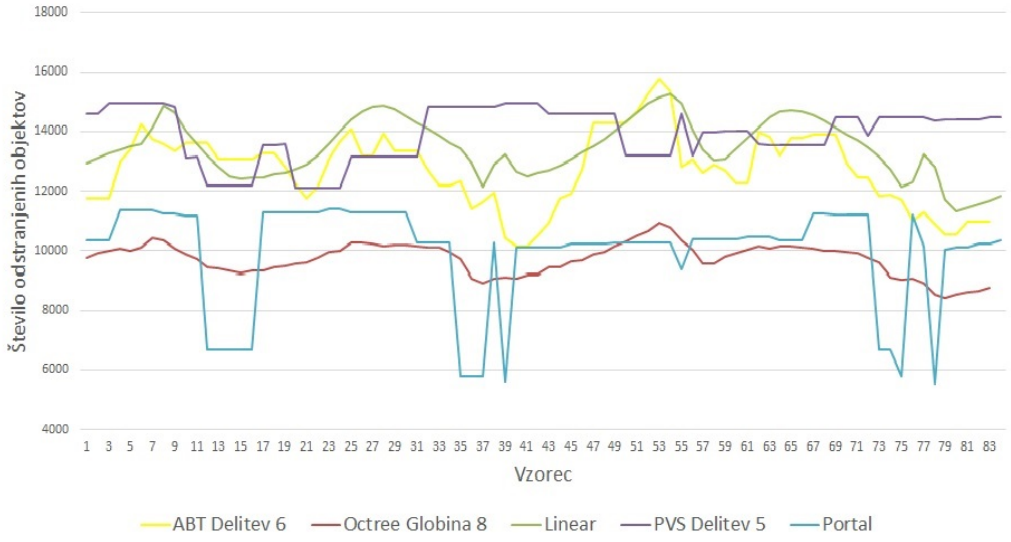
5.4.6 Primerjava

5.4.6.1 Odstranjevanje objektov

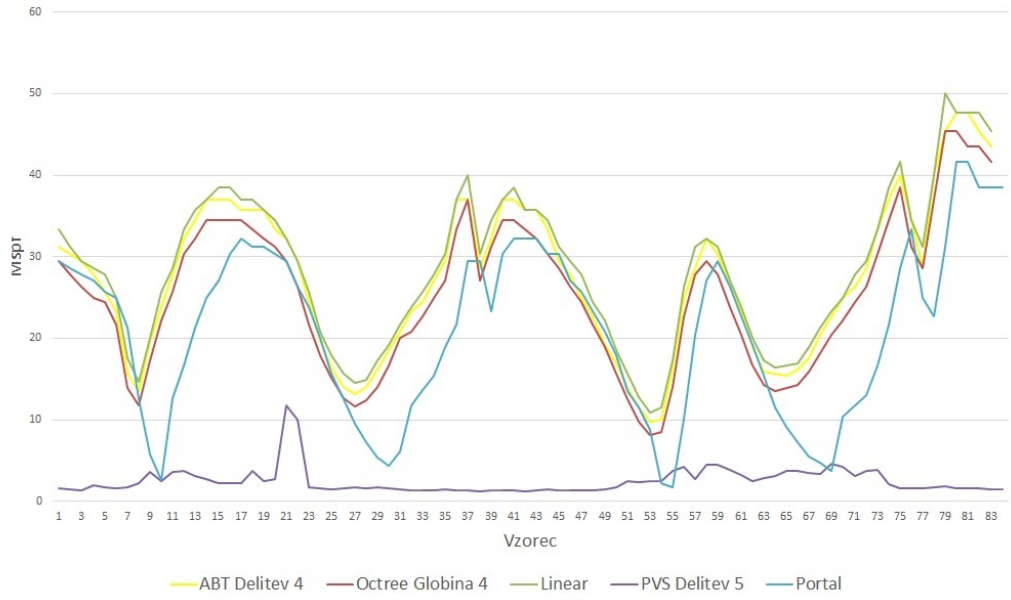
Prvo mesto v odstranjevanju si delita graf PVS in linearen seznam (slika 5.66). PVS izkoristi veliko prekrivanje objektov, ki se pojavi v tem tipu sveta, saj se scenarij odvija znotraj stavbe. Zelo blizu je drevo ABT, ki krepko prekosi Octree, in sicer za več kot 3000-4000 objektov tekom odvijanja scenarija. Tu je treba omeniti, da se število odstranjenih objektov precej poveča zaradi novo nastalih objektov, ki nastanejo pri ABT delitvah. Najslabše se odreže sistem portalov, ki ima med drugim tudi najgloblje padce. Pri vseh grafih razen PVS se opazi vpliv položaja kamere v scenariju. Manjše odstranitve se namreč pojavijo, ko kamera opazuje zunanji svet.

5.4.6.2 Performančnost

Hitrost vseh grafov je v veliki meri odvisna od števila odstranjenih objektov, razen pri PVS, ki premaga vse oblike grafov. Razlika znaša od 5 do 30 milisekund. Pri binarnih razdelitvah prostora Octree drevesu uspe premagati ABT v vseh vzorcih, kar je posledica predvsem prevelikega ustvarjanja novih objektov pri ABT drevesu. Portali so najslabši pri odstranitvah, vendar po performančnosti najboljši med vsem, z izjemo PVS-ja (slika 5.67). Sistem portalov je kljub slabši performančnosti še vedno boljša izbira kot PVS. Napak pri upodabljanju ni, saj se portali zelo lepo prilagajajo pol-odprtemu prostoru, medtem ko je PVS zelo občutljiv na uporabljeno delitev.



Slika 5.66: Število odstranjenih objektov najboljših predstavnikov v pol-odprtem svetu.



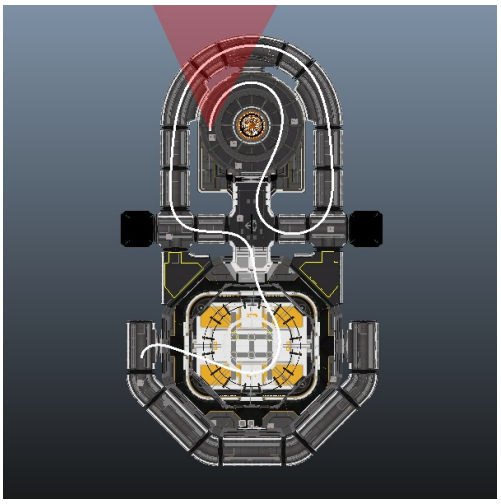
Slika 5.67: Mspf performančnost najboljših predstavnikov v pol-odprtem svetu.

5.5 Zaprti svet

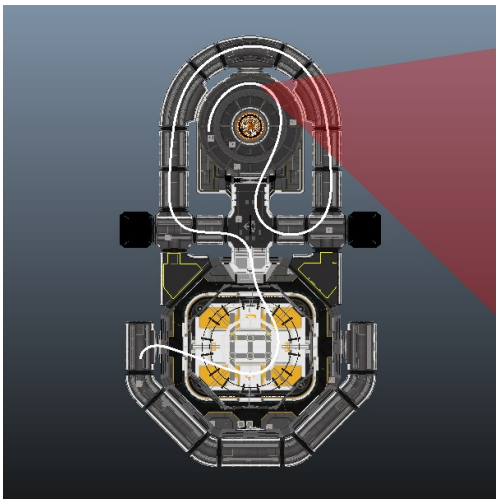
5.5.1 Linearen seznam

5.5.1.1 Odstranjevanje objektov

Število odstranitev je odvisno v večji meri od tega, koliko palet s kockami zajame vidno polje kamere, in v manjši meri od objektov, ki sestavljajo laboratorij. Pri vzorcu 1 (slika 5.68) ni vidna nobena izmed palet kock, saj je kamera postavljena na rob sveta s pogledom stran od središča. Število odstranjenih objektov je okoli 15000, kar predstavlja večino sveta. Pri vzorcu 6 (slika 5.69) število odstranitev začne padati, ko kamera prvič zajame palete kock in se obrača proti središču. Minimum je dosežen pri vzorcu 9 (slika 5.70), ki predstavlja tudi globalni minimum celotnega scenarija. Čim kamera naredi zavoj stran od središča sveta, se odstranitve strmo povečajo (slika 5.71).

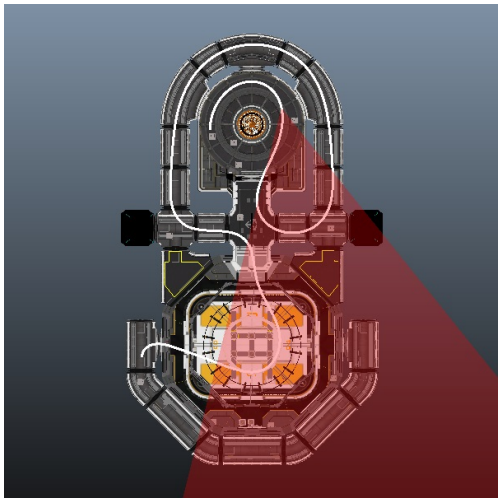


Slika 5.68: Vzorec 1, linearen seznam, zaprti svet.

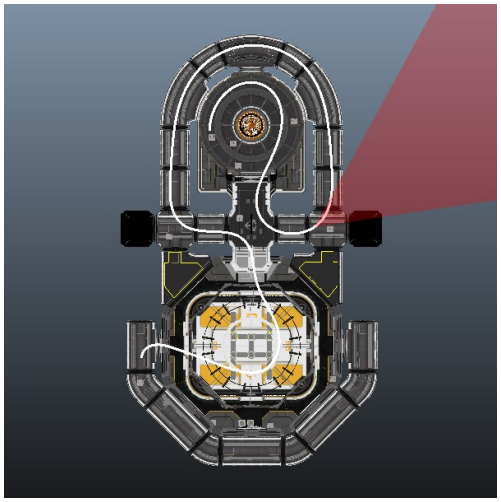


Slika 5.69: Vzorec 6, linearen seznam, zaprti svet.

Potek scenarija naredi lok po koridorju, kjer odstranitve ostajajo na maksimalni višini zaradi položaja in kota kamera. Naslednji padec je zabeležen pri vzorcu 44 (slika 5.72) s pogledom kamere proti središču sveta in večjim številom palet v vidnem polju. Padec se ustali pri vzorcu 54 (slika 5.73) z največjim številom palet kock v vidnem polju. Od tega trenutka dalje se kamera nahaja na robu sveta,

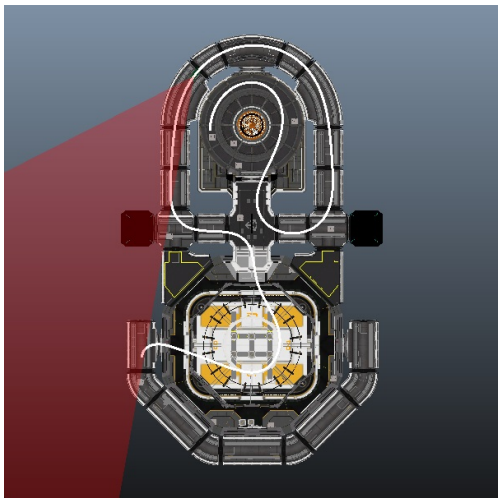


Slika 5.70: Vzorec 9, linearen seznam, zaprti svet.

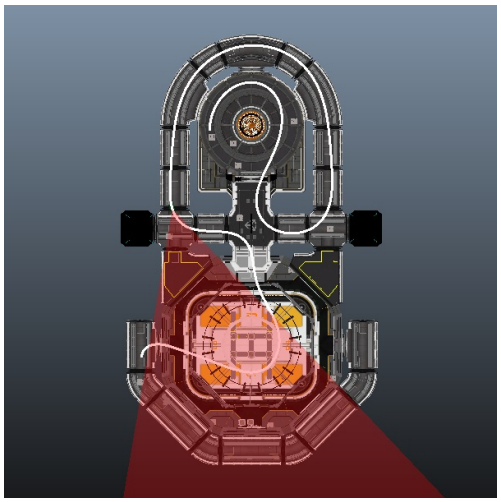


Slika 5.71: Vzorec 23, linearen seznam, zaprti svet.

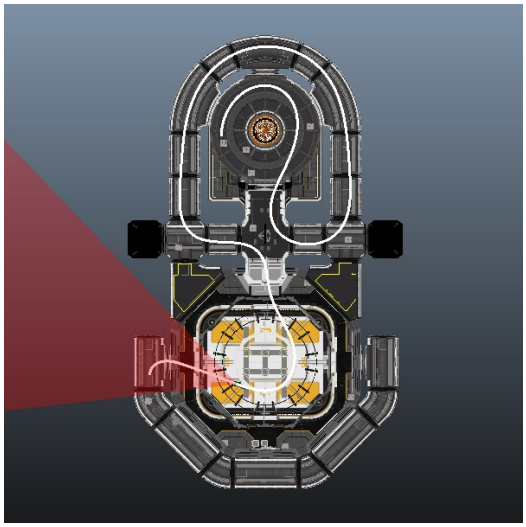
s pogledom navzven, in odstranitve ponovno začnejo naraščati do maksimalnega nivoja pri vzorcu 77 (slika 5.74). Količina odstranitev med potovanjem kamere je vidna na sliki 5.75.



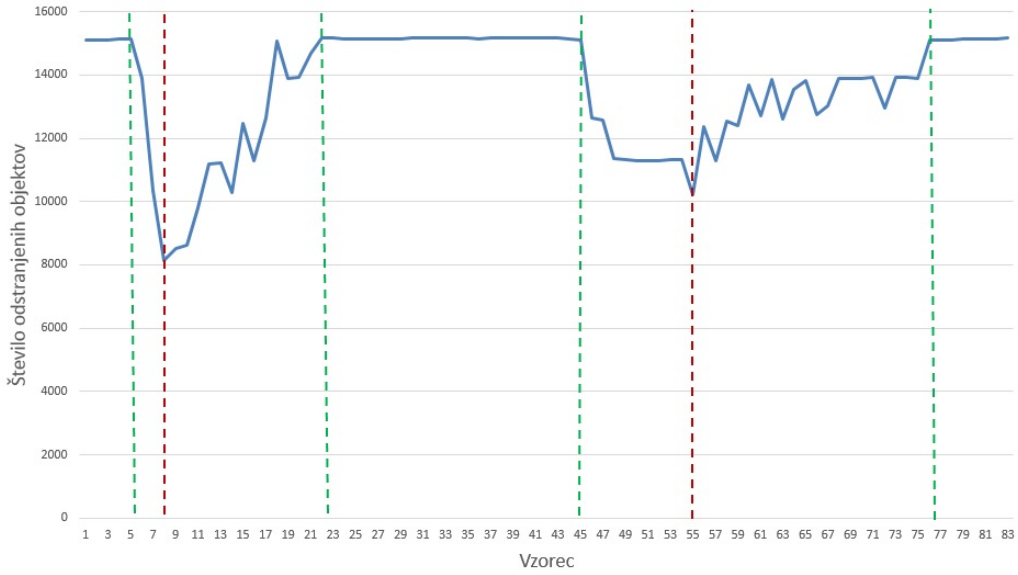
Slika 5.72: Vzorec 44, linearen seznam, zaprti svet.



Slika 5.73: Vzorec 54, linearen seznam, zaprti svet.



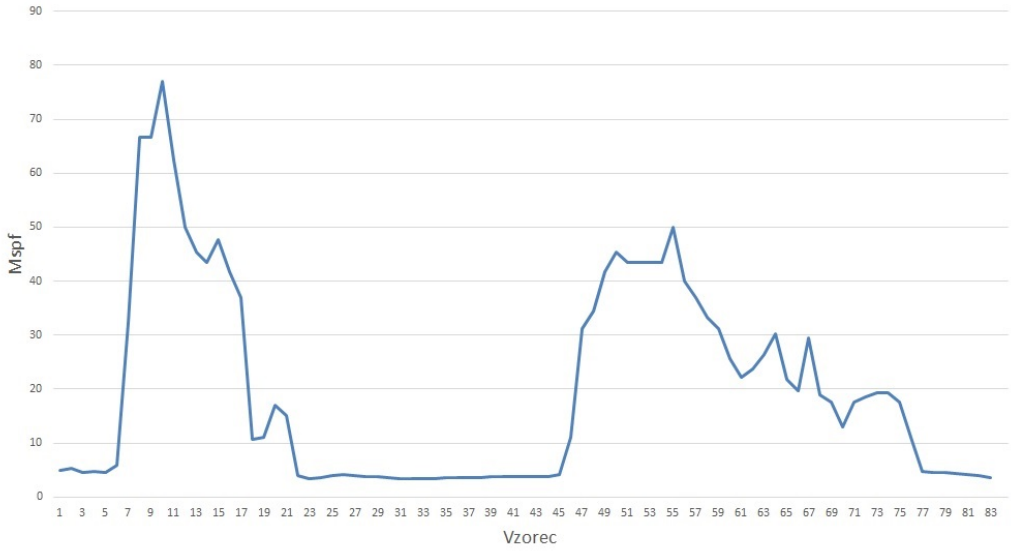
Slika 5.74: Vzorec 77, linearen seznam, zaprti svet.



Slika 5.75: Število odstranjenih objektov z linearnim seznamom v zaprtem svetu.

5.5.1.2 Performančnost

Hitrost seznama (slika 5.76) doživlja ogromne skoke v performančnosti. Performančnost je neposredno odvisna od količine odstranjenih objektov, saj preostanek objektov obremenjuje grafično enoto in negativno vpliva na hitrost. V najboljšem primeru je pri intervalih vzorcev 0-8, 23-43 in 77-83 hitrost zelo ugodna, in sicer okoli 5 milisekund, v najslabšem primeru pa doseže celo 78 milisekund (pri vzorcu 10).



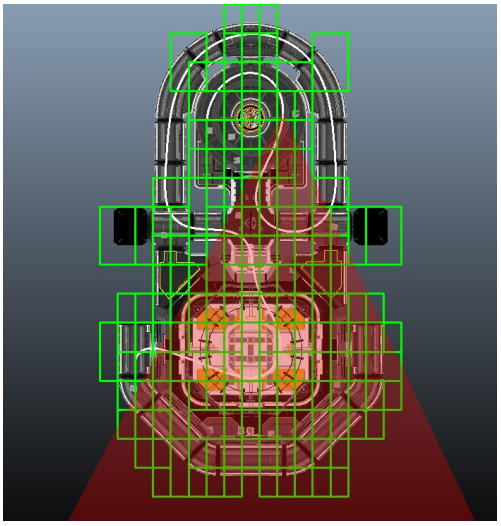
Slika 5.76: Mspf performančnost linearnega seznama v zaprtem svetu.

5.5.2 Octree

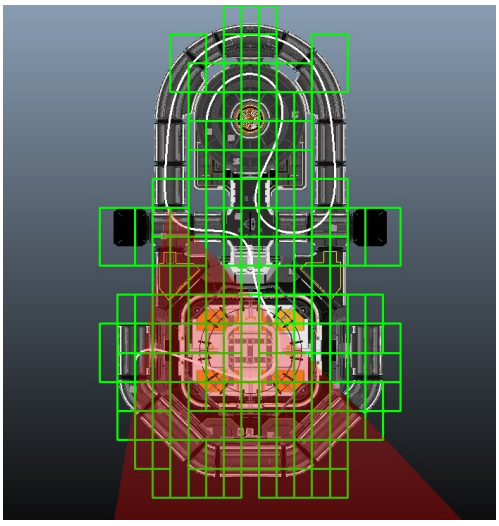
5.5.2.1 Odstranjevanje objektov

Octree drevo ni odvisno od oblike sveta temveč od strukture, kar pomeni, da bo količina odstranjevanja odvisna predvsem od tega, kako bo svet zgrajen, ne glede na to, ali bo to svet odprtega, pol-odprtega ali zaprtega tipa. Ne glede na globino doživi število odstranitev pri vzorcih 9 in 55 (sliki 5.77, 5.78) dva občutna padca z minimumi, in sicer ko se kamera obrača proti središču sveta. Na slikah 5.77 in 5.78 so oktanti globine 4 označeni s svetlozeleno barvo. Glede odstranjevanja objektov na nivoju grafa je najbolj očitna posebnost UnityLab sveta precejšna

podobnost med globinami 4, 6 in 8, ki so bile v prejšnjih svetovih prikazovale večino poteka scenarija zelo različne rezultate. Izjema je vzorec 9, kjer je razlika v količini odstranjenih objektov okoli 4000. Na tem mestu dodatno drobljenje oktantov zelo izboljša rezultat, saj palete kock vsebuje veliko število objektov. Količina odstranitev med izvajanjem scenarija je prikazana na sliki 5.80.



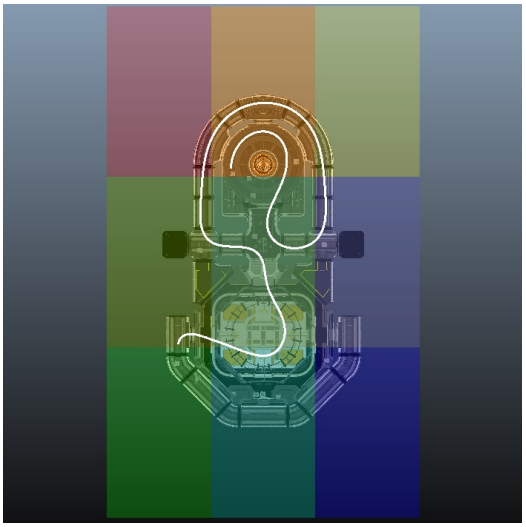
Slika 5.77: Vzorec 9 pri Octree globine 4 v zaprtem svetu.



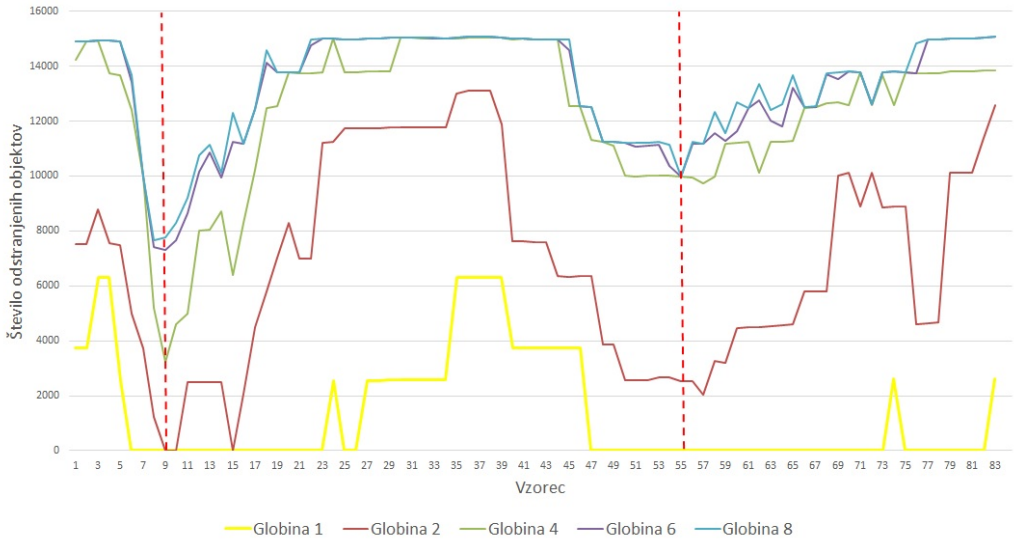
Slika 5.78: Vzorec 54 pri Octree globine 4 v zaprtem svetu.

V prejšnjih svetovih je pri ohlapnih Octree drevesih globina 1 slabo omogočala odstranjevanje. Pri takšnem poteku scenarija pa je odstranjevanje možno tudi na nivoju 1, saj pot kamere obstaja tudi izven presečišča ohlapnih oktantov globine 1. Na sliki 5.79 je pot scenarija označena z belo barvo, oktanti pa so obarvani z rdečo, rumeno, zeleno in modro barvo. Ko pride do prekrivanja področij, pa so uporabljene mešane barve.

Število vidnih celic pri globinah 4, 6 in 8 se na področjih minimumov občutno poveča. Povečanje je bistveno večje pri vzorcu 9 kot pri 54 (slika 5.83). Pri obeh vzorcih je v vidnem polju kamere precejšen del sveta in je posledično vidnih tudi več oktantov. Na sliki 5.81 in 5.82 so prikazani oktanti, ki se nahajajo na globini 6 in 8. Označeni so s svetlo zeleno in roza barvo. Zaradi pogleda navzdol slika ne razkriva velikega števila oktantov, razporejenih po y osi, kar pa je močno opazno pri številu vidnih oktantov na vzorec (sliki 5.83).



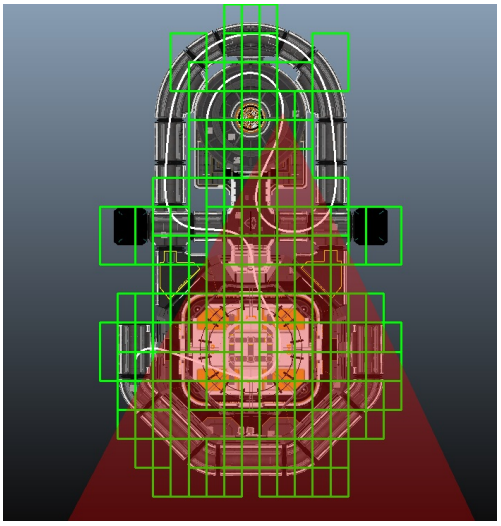
Slika 5.79: Oktanti ohlapnega Octree drevesa globine 1 v zaprtem svetu.



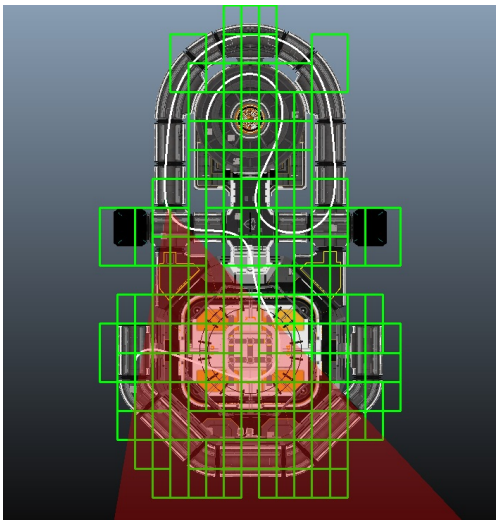
Slika 5.80: Število odstranjenih objektov z Octree drevesom v zaprtem svetu.

5.5.2.2 Performančnost

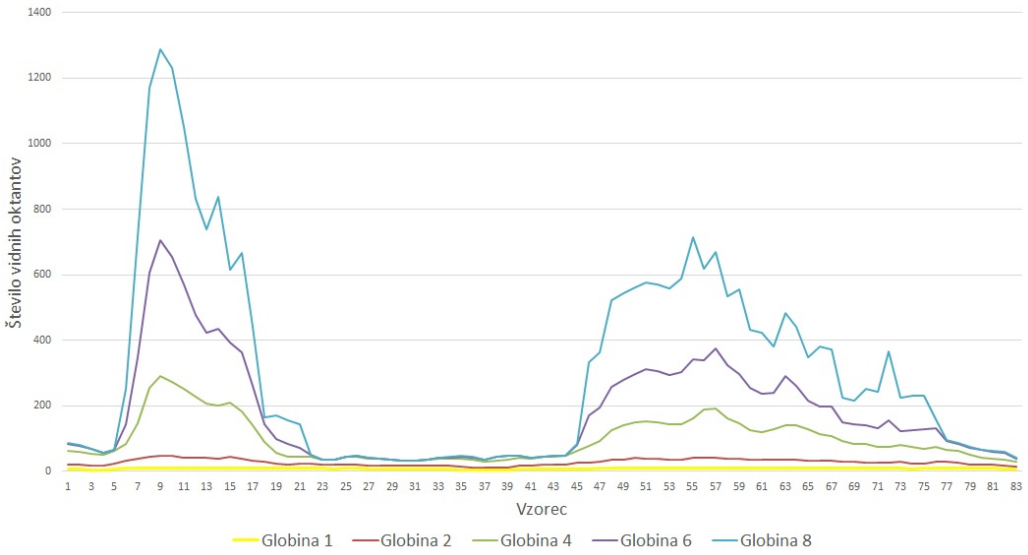
Vse globine Octree drevesa so si po performančnosti pri položajih v scenarijih zelo blizu, še posebej ko je število odstranjevanje nizko. Največja prednost višjih globin je hitrost, ko je kamera obrnjena stran od središča sveta, in sicer na intervalih 1-5, 23-44 in 77-83 (slika 5.80). Najboljši rezultati so pridobljeni z globino 4, ki ji je



Slika 5.81: Vzorec 9 pri Octree v zaprtem svetu, globine 6, 7, 8.



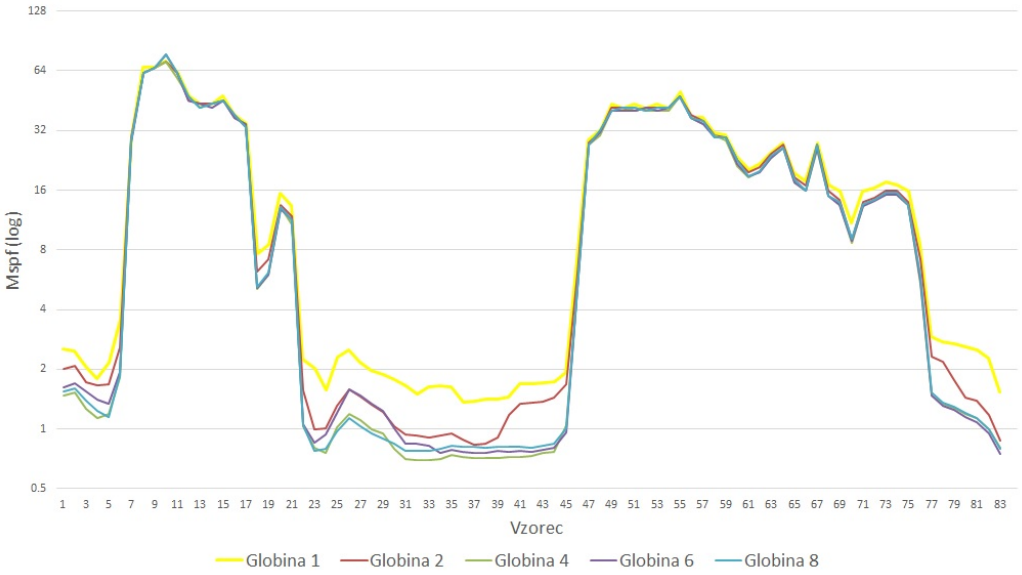
Slika 5.82: Vzorec 54 pri Octree v zaprtem svetu, globine 6, 7, 8.



Slika 5.83: Število vidnih oktantov Octree drevesa v zaprtem svetu.

zelo blizu globina 8. V prejšnjih svetovih je globina 8 kvečjemu negativno vplivala na rezultat. Razlika se skriva v dejstvu, da je zapolnjenost oktantov veliko boljša, kar se opazi v razliki števila odstranitev na vzorcu 9 in 54 (slika 5.84). Ta razlika nadoknadi izgubljeno performančnost zaradi večjega procesiranja na nivoju grafa

z občutno manjšim procesiranjem na nivoju kamere.

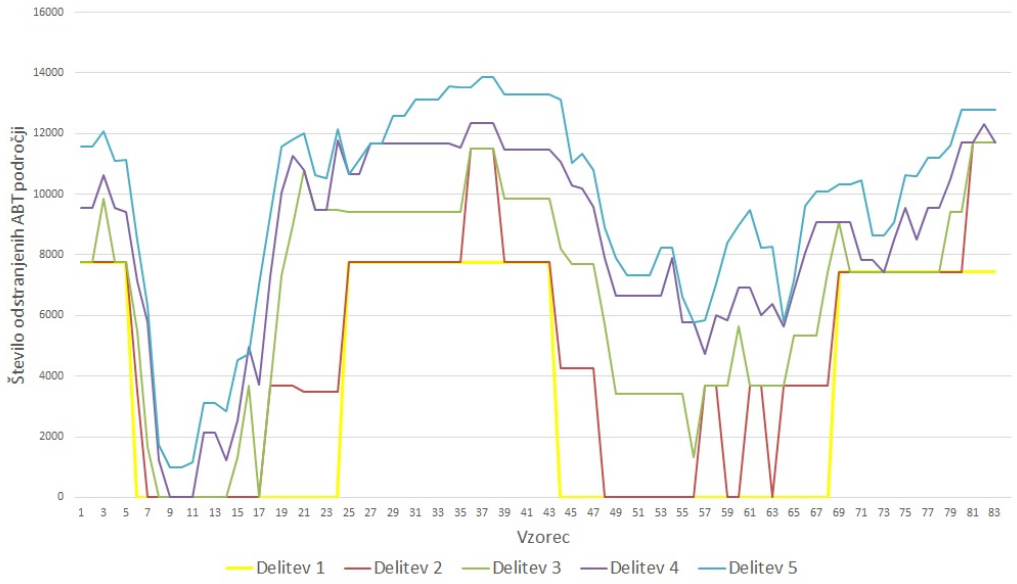


Slika 5.84: Mspf performančnost Octree drevesa v zaprtem svetu.

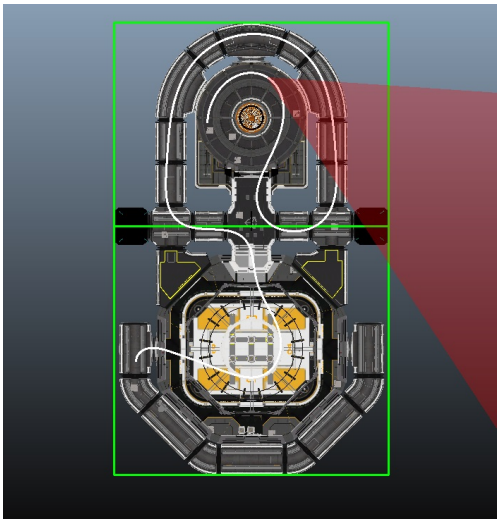
5.5.3 ABT

5.5.3.1 Odstranjevanje objektov

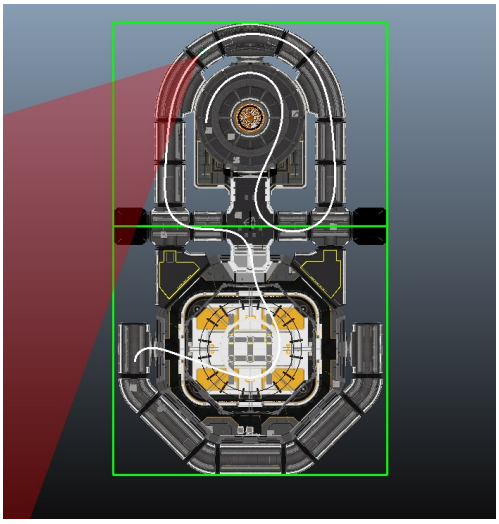
Količina odstranitev (slika 5.85) narašča z večanjem delitev ABT drevesa. Najboljši rezultati so bili doseženi pri 5-kratni delitvi, najslabši pa pri 1-kratni delitvi. Pri vseh delitvah se pojavijo minimumi pri vzorcih 10 in 57 ter maksimumi pri vzorcih 7, 36 in 83. Najmanjša odstranjevanja so posledica usmeritve in položaja kamere, ki je v teh primerih obrnjena proti središču, nahaja pa se na robu sveta s pogledom skozi celoten svet. V okolici vzorcev 10 in 57 ABT drevo pri 1- ter 2-kratnih delitvah popolnoma odpove, saj se odstranitve zmanjšajo na nič (slike 5.86, 5.87, 5.88, 5.89). Pri 5-kratni delitvi so največji maksimumi zabeleženi pri vzorcih 5 in 37 (sliki 5.90, 5.91) ter na koncu scenarija pri vzorcu 83.



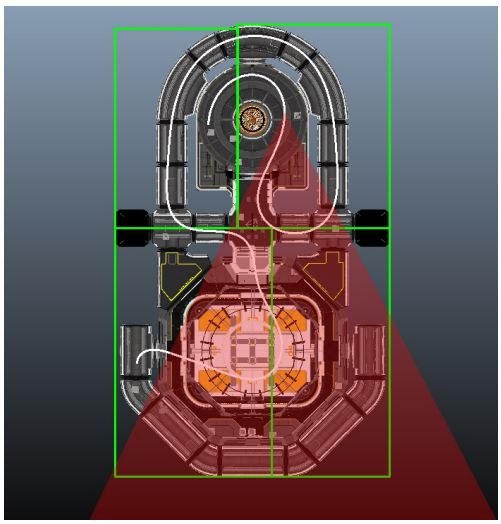
Slika 5.85: Število odstranjenih objektov z ABT drevesom v zaprtem svetu.



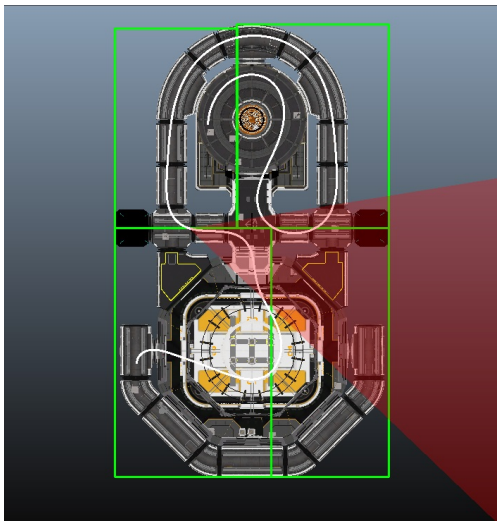
Slika 5.86: Vzorec 9 pri ABT 1-kratni delitvi v zaprtem svetu.



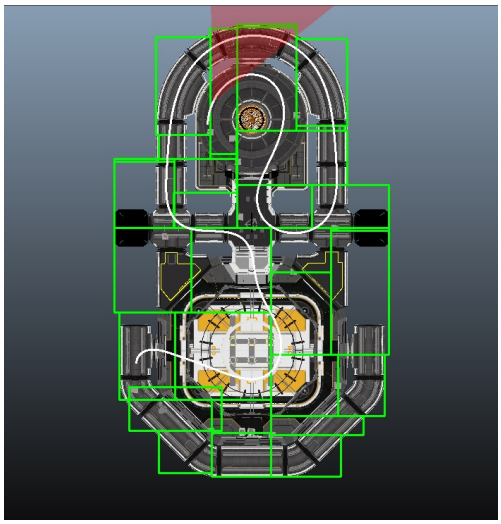
Slika 5.87: Vzorec 54 pri ABT 1-kratni delitvi v zaprtem svetu.



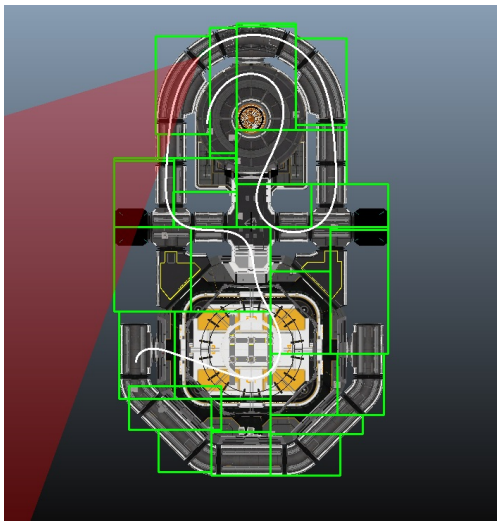
Slika 5.88: Vzorec 9 pri ABT 2-kratni delitvi v zaprtem svetu.



Slika 5.89: Vzorec 54 pri ABT 2-kratni delitvi v zaprtem svetu.



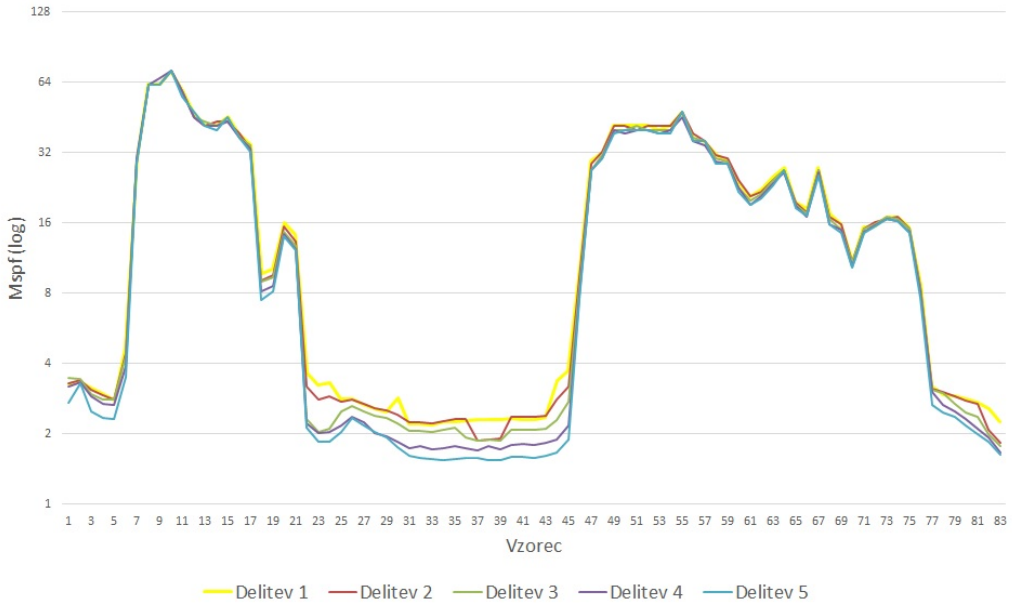
Slika 5.90: Vzorec 5 pri ABT 5-kratni delitvi v zaprtem svetu.



Slika 5.91: Vzorec 37 pri ABT 5-kratni delitvi v zaprtem svetu.

5.5.3.2 Performančnost

Povečevanje globine postopoma zvišuje tudi hitrost, kar je najbolj opazno v odsekih scenarija, kjer je največ odstranitev (slika 5.92). Pri teh okvirjih lahko znaša razlika do 5 milisekund. Hitrost na preostalem delu scenarija ostaja dokaj podobna, vendar je še vedno opazna zadostna razlika okoli 1-2 milisekunde. Povečevanje globine ima pozitiven učinek, kar pomeni, da so zadostno zapolnjena novo nastala področja višjih nivojev in je le malo novih ustvarjenih objektov.



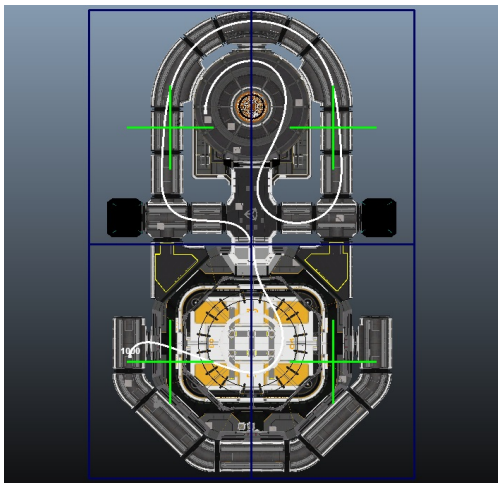
Slika 5.92: Mspf performančnost ABT drevesa v zaprtem svetu.

5.5.4 PVS

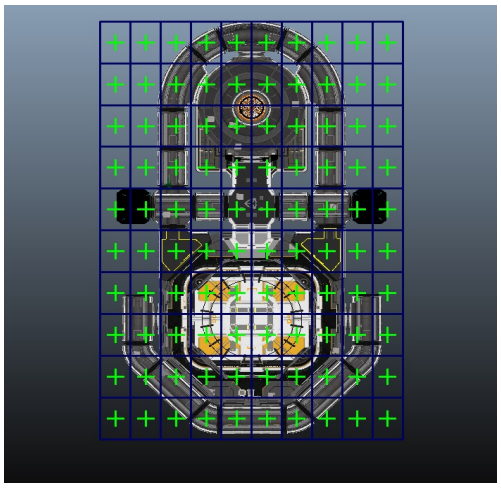
5.5.4.1 Odstranjevanje objektov

Odstranjevanje objektov s PVS grafom se najboljše odreže v zaprtem svetu, kjer je obilo priložnosti za prekrivanje objektov. Z zelo malo padci je najbolj konstantna 10-kratna delitev, ki dobro zajema strukturo laboratorija. Največje spremembe v odstranjevanju se pojavljajo pri 2-kratni delitvi. Na začetku izvajanja scenarija se kamera nahaja v zgornji desni celici, kjer se nahaja odprti prostor. Prehod v celico, kjer se nahaja koridor, je zelo omejil kamero pri grajenju PVS grafa in

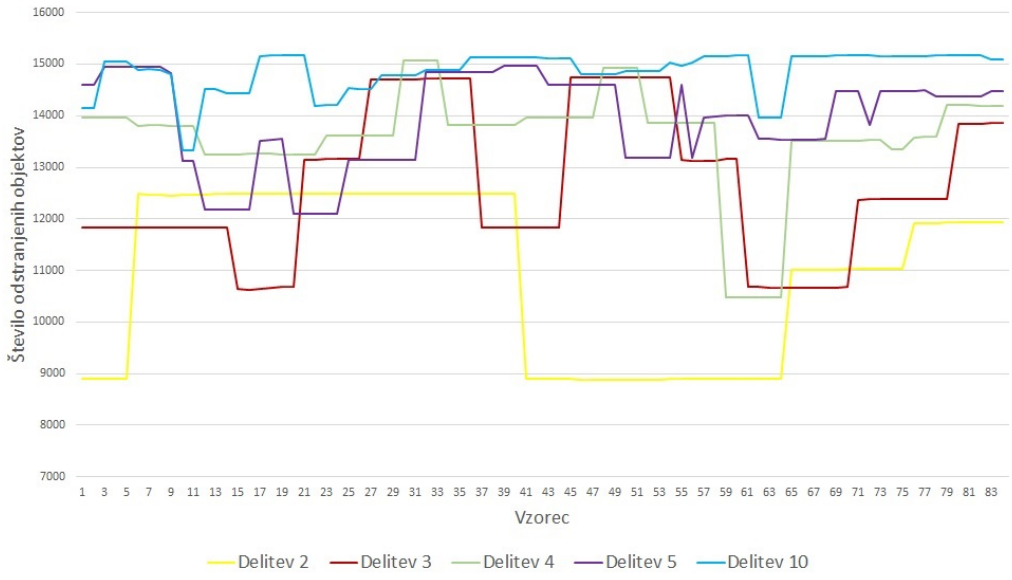
je zato temu primerno večje odstranjevanje. Z večanjem delitev dosežemo vedno boljše prileganje in boljše odstranjevanje (slika 5.95). Na sliki 5.93 in sliki 5.94 je primer PVS grafa z delitvama 2 in 3, kjer zeleni križci predstavljajo lokacijo kamere pri gradnji PVS grafa, celice pa so označene s temno modro barvo.



Slika 5.93: PVS graf 2-kratne delitve v zaprtem svetu.



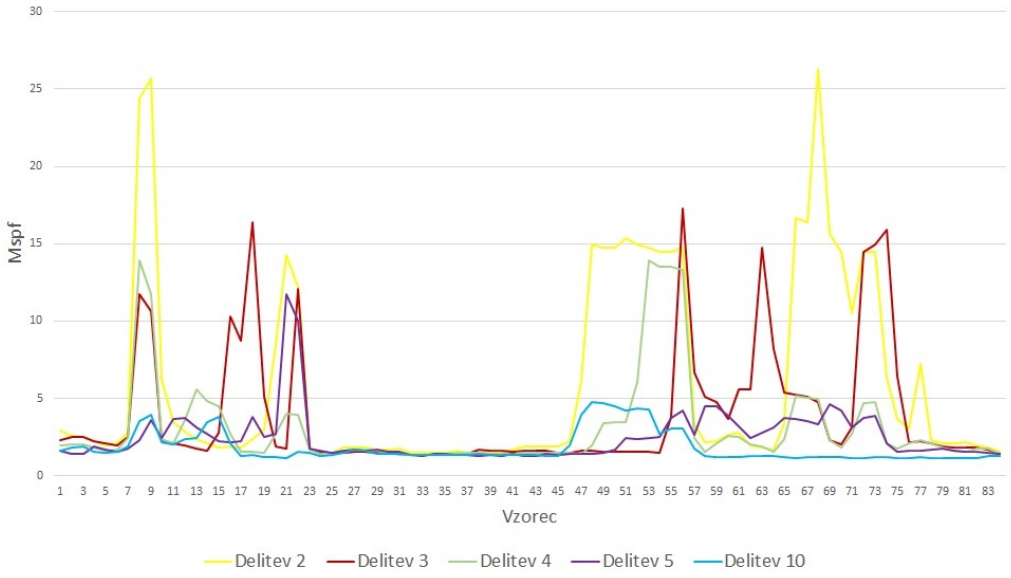
Slika 5.94: PVS graf 10-kratne delitve v zaprtem svetu.



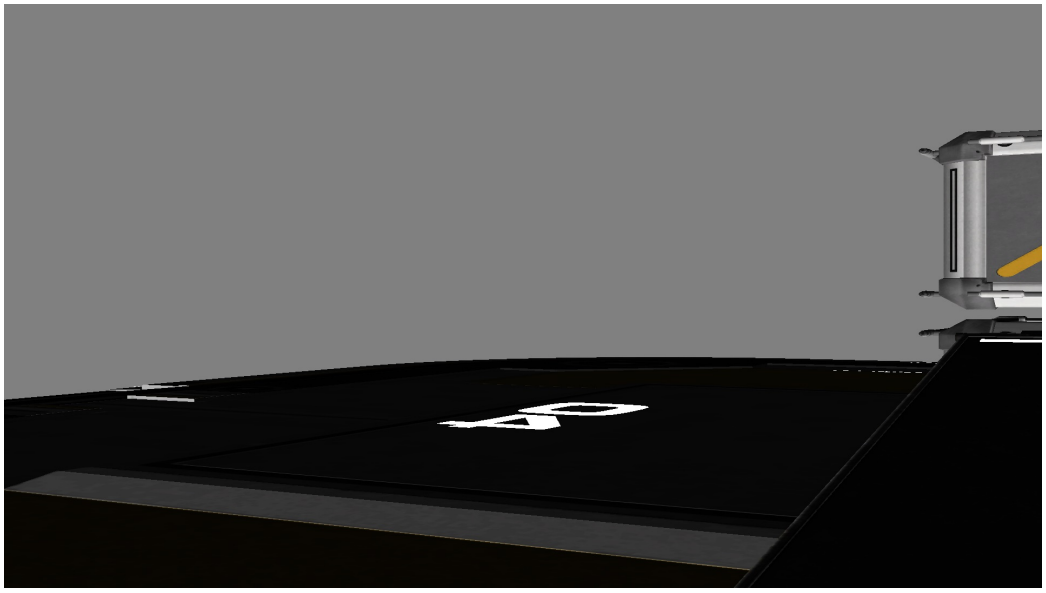
Slika 5.95: Število odstranjenih objektov s PVS grafom v zaprtem svetu.

5.5.4.2 Performančnost

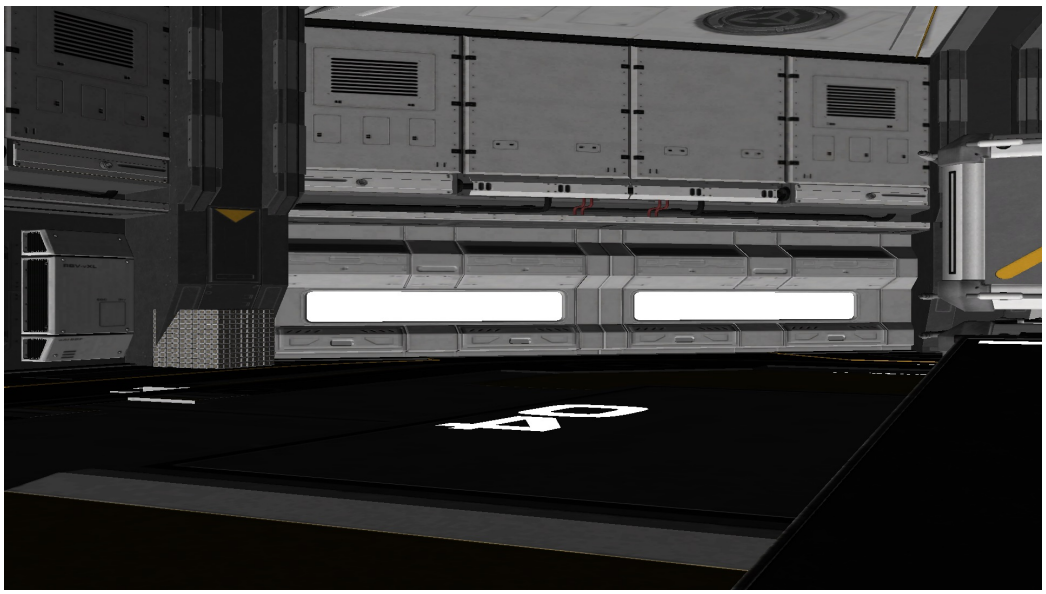
Absolutni zmagovalec performančnih testov je 10-kratna delitev, predvsem zaradi dobrega prileganja in velikega odstranjevanja. Najslabši rezultati so pridobljeni z 2-kratno delitvijo, ki tudi najmanj izkorišča prekrivanje (slika 5.96). Konsistenca upodabljanja je prav tako najboljša pri 10-kratni delitvi. Na slikah 5.97 in 5.98 je prikazan primer nepravilnega in pravilnega upodabljanja za 2- in 10-kratno delitev.



Slika 5.96: Mspf performančnost PVS grafa v zaprtem svetu.



Slika 5.97: Nepravilnost upodabljana pri PVS grafu delitve 2 v zaprtem svetu (glej slika 5.98).

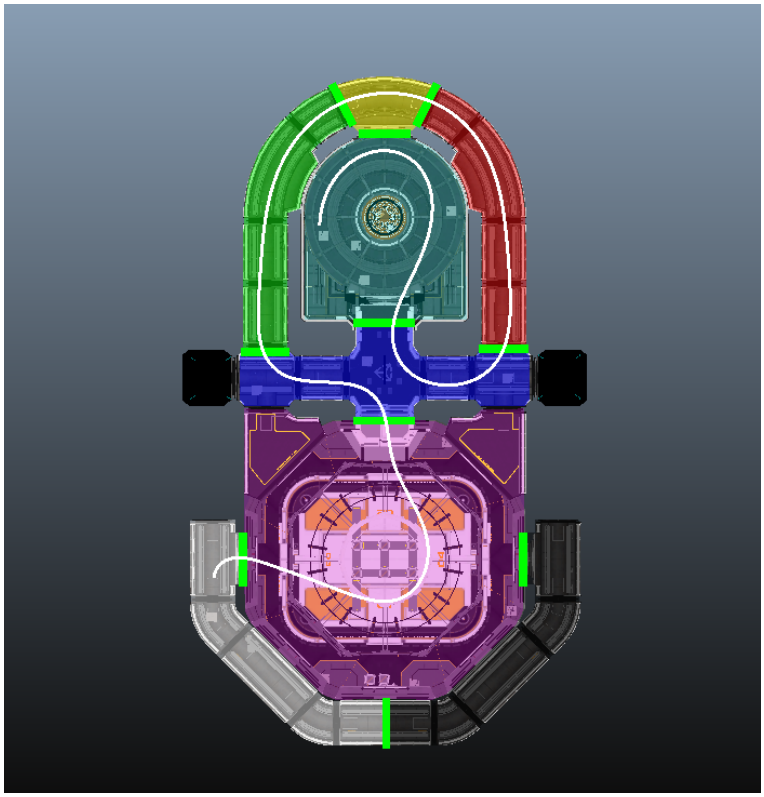


Slika 5.98: Pravilnost upodabljana pri PVS grafu delitve 10 v zaprtem svetu.

5.5.5 Portali

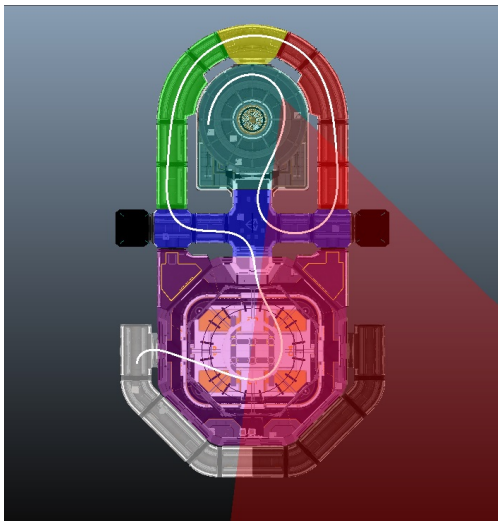
5.5.5.1 Odstranjevanje objektov

Tekom izvajanja scenarija kamera prepotuje sedem s portali omejenih področij, ki so na sliki 5.99 prikazana vsako s svojo barvo, medtem ko odebeljene zelene črte predstavljajo dejanske portale oziroma prehode. Odstranjevanje objektov na nivoju grafa je neposredno povezano s številom vidnih portalov v določenem okvirju in v primerjavi z ostalimi grafi ni toliko odvisno od dejstva, ali kamera gleda proti središču sveta (slika 5.104).

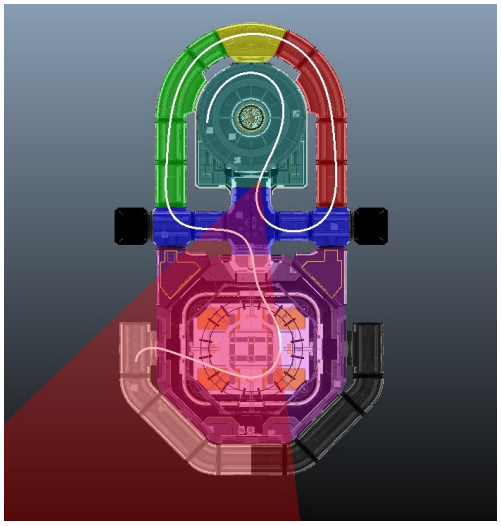


Slika 5.99: Cone in postavitev portalov v zaprtem svetu.

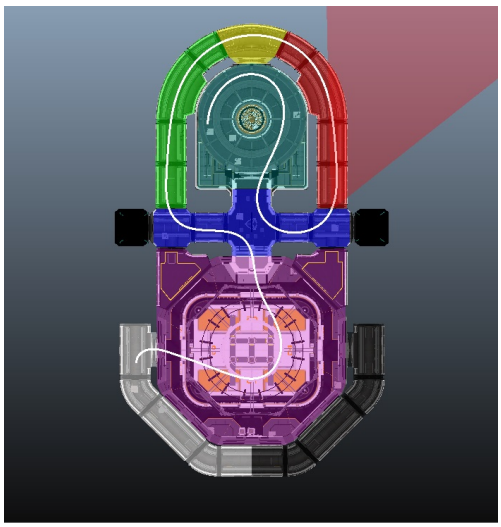
Najbolj občuten padec je na intervalu okvirjev od 8-18 (sliki 5.100, 5.101), kjer je vidnih največ portalov in so posledično odstranitve najmanj številčne. Največje vrednosti se dosežejo pri prehodih kamere na stranske loke zgradbe, ker je vidni kot kamere najbolj omejen (sliki 5.102, 5.103).



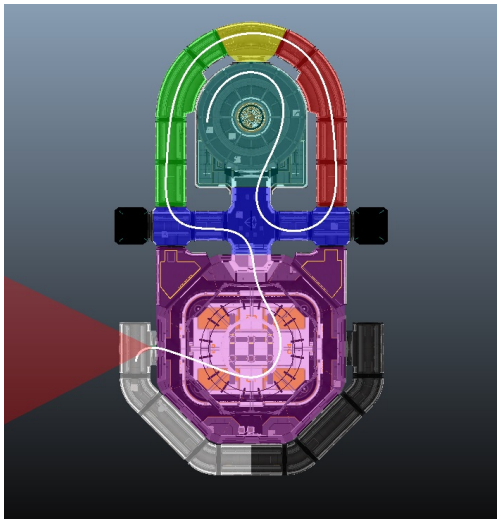
Slika 5.100: Vzorec 8 pri portalih v zaprtem svetu.



Slika 5.101: Vzorec 18 pri portalih v zaprtem svetu.



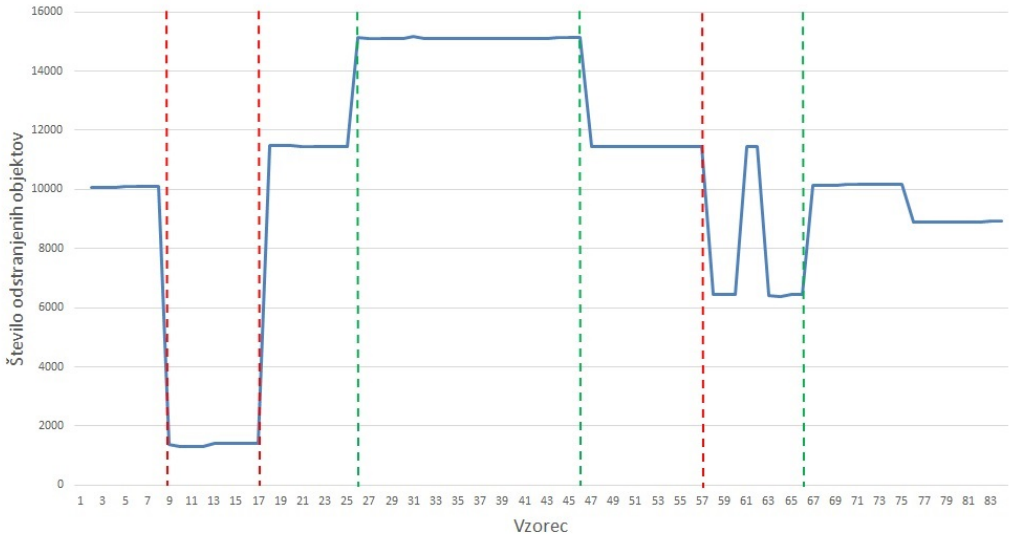
Slika 5.102: Vzorec 27 pri portalih v zaprtem svetu.



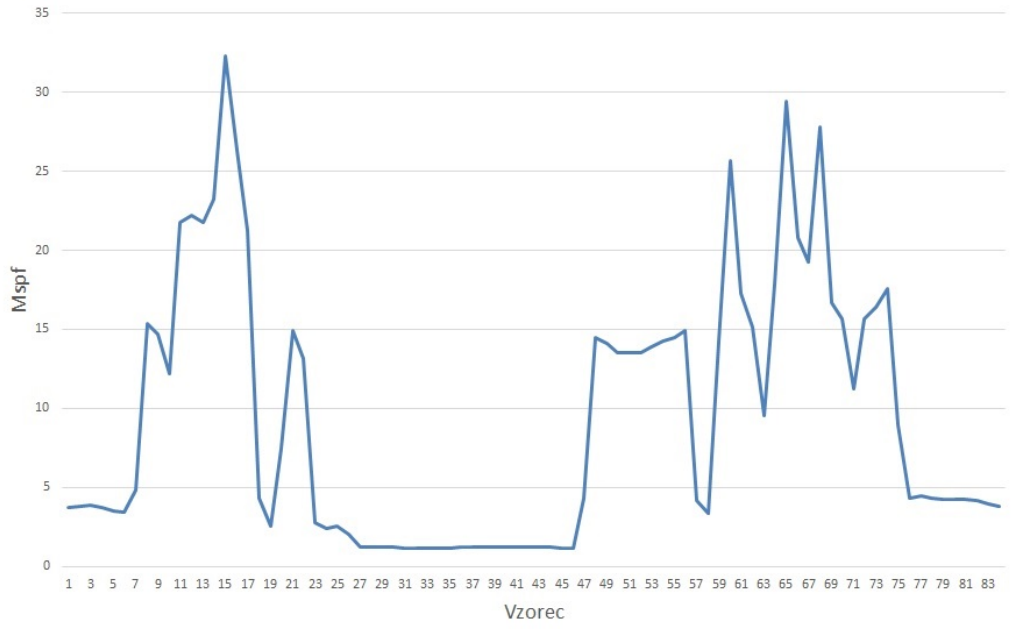
Slika 5.103: Vzorec 77 pri portalih v zaprtem svetu.

5.5.5.2 Performančnost

Analiza performančnosti (slika 5.105) nakazuje vpliv števila odstranitev na nivoju grafa, ki pa ne sledi jasnemu vzorcu. Povečevanje števila odstranitev ugodno vpliva na hitrost, vendar je ta vpliv vse prej kot linearen.



Slika 5.104: Število odstranjenih objektov s portali v zaprtem svetu.

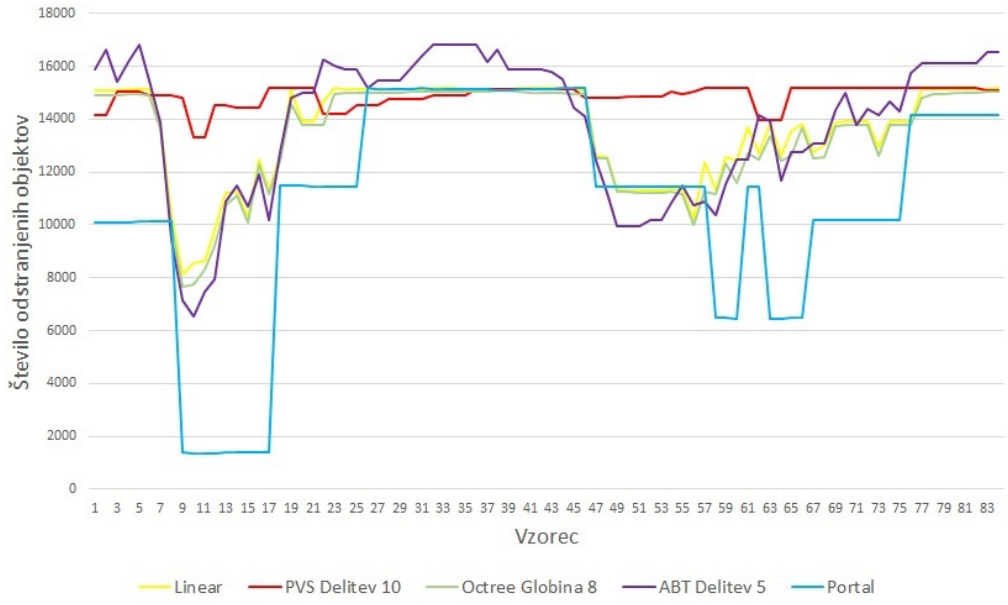


Slika 5.105: Mspf performančnost portalov v zaprtem svetu.

5.5.6 Primerjava

5.5.6.1 Odstranjevanje objektov

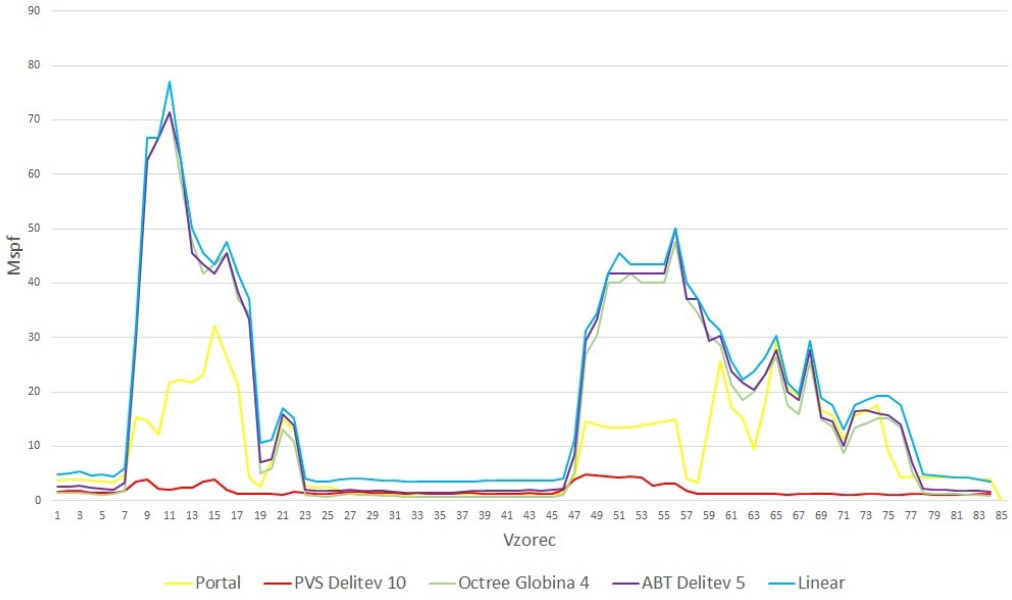
Ponovno je najboljši v odstranjevanju PVS graf, kar je predvsem posledica zaprtega notranjega sveta. V primerjavi z ostalimi grafi je prav tako precej neobčutljiv na položaj kamere. Tako kot linearen seznam tudi Octree odstrani skoraj isto količino objektov in tesno sledi grafu ABT. Izmed vseh grafov ima sistem portalov najslabše rezultate. Vsi grafi, z izjemo PVS, so občutljivi na vidno polje in položaj kamere. Rezultat sistema portalov bi se dalo občutno izboljšati z drugačno postavitvijo portalov, ki zaradi zaprte narave sveta nikakor ne bi omejevala pot kamere (slika 5.106).



Slika 5.106: Število odstranjenih objektov z najboljšimi predstavniki v zaprtem svetu.

5.5.6.2 Performančnost

Pri performančnosti se linearen seznam izkaže za najslabšega. Sledita mu Octree in ABT, ki delovanje izboljšata okvirno do 5 milisekund. Najboljša grafa sta PVS in sistem portalov, kjer portali znova dokažejo, da količina odstranitev na nivoju grafa ni nujno najpomembnejša metrika (slika 5.107).



Slika 5.107: Mspf performančnost najboljših predstavnikov v zaprtem svetu.

Poglavje 6

Zaključek

V diplomskem delu je bilo obravnavano obnašanje grafov scene različnih oblik svetov, ki se pojavljajo v modernih video igrah. Za potrebe testiranja grafov scen je bila razvita testna aplikacija, ki uporablja odprto kodni pogon Ogre3D. Aplikacija je tesno povezana z orodjem Maya, s katerim je bil opravljen izvoz svetov v Ogre3D. Za vsako obliko sveta in za vsak graf scene so bili izvršeni scenariji testiranja, ki popeljejo kamero oziroma igralca po vnaprej določeni krivulji. Med potovanjem kamere so bile zajete različne metrike, ki nudijo vpogled v efektivnost delovanja grafa scene in njegovo primernost za podano obliko sveta.

Vse pridobljene ugotovitve temeljijo na predpostavki, da so svetovi statični, kar izključuje prisotnost premikajočih objektov. Poleg odstranjevanja na nivoju grafa uporablja vsak graf scene kot končni korak še odstranjevanje na podlagi vidnega polja kamere. Z vidika performančnosti se je najbolje odrezal PVS graf, ki ne glede na obliko sveta krepko premaga vse tekmece. PVS graf je tudi edini graf scene, ki upošteva prekrivanje objektov pri določanju vidnosti. Po količini odstranjenih objektov je prav tako eden izmed najboljših, razen pri odprtem tipu sveta. Kljub temu se izkaže, da je preprostost PVS algoritma ključnega pomena, saj algoritem še vedno dosega boljšo performančnost. Edina slabost algoritma je občutljivost na število izbranih vzorčnih točk, kar se izkazuje pri napakah v upodabljanju ter večji pomnilniški zahtevnosti. Povečevanje vzorčnih točk zelo podaljša čas izgradnje PVS grafa, ki tako lahko znaša več ur. Glede na to da hrani PVS graf za vsako celico seznam objektov, se v skladu s tem poveča tudi potreben prostor v pomnilniku.

Sistem portalov je po performančnosti drugo uvrščeni algoritem. Pri portalih je zanimivo dejstvo, da se glede odstranjevanja na nivoju grafa v večini primerov uvrsti med najslabše, vendar je po performančnosti skoraj vedno drugo uvrščen. Na podlagi teh dveh ugotovitev je moč sklepati, da je odstranjevanje na nivoju kamere zelo hitro in ima velik vpliv, le ko gre za veliko količino objektov. Sama implementacija sistema portalov pa je veliko hitrejša kot pri binarnih razdelitvah.

Najslabši rezultati so bili pridobljeni pri pol-odprtem svetu, kjer je sistem portalov po performančnosti blizu Octree in ABT. V tej obliki sveta sistem portalov dosega zelo nizke vrednosti odstranitev in se posledično opazi povečan čas odstranjevanja na nivoju kamere. Velja omeniti, da je čas potreben za odstranjevanje s kamero odvisen od položaja in obsega objektov. Če se objekt nahaja popolnoma izven vidnega polja kamere, je za določitev vidnosti potrebno manj računske moči, kot če se objekt nahaja znotraj vidnega polja. Po večini predstavlja razlika v številu odstranitev na nivoju grafa med portali in recimo Octree ravno objekte izven polja. Upoštevajoč prejšnjo ugotovitev in dejstvo, da je število con pri portalih veliko manjše kot pa število oktantov pri Octree, je tako mogoče pojasniti izboljšanje performančnosti.

Algoritmi binarnega razdeljevanja so glede performačnosti srednje uvrščeni. Čeprav ti algoritmi skoraj najboljše odstranjujejo na nivoju grafa, to očitno ne vpliva na njihovo performančnost. Nekateri izmed možnih razlogov so bili navedeni že v prejšnjih odstavkih, vendar je potrebno pri binarnih razdelitvah omeniti še sledeče: Octree in ABT drevesi imata najbolj razvejano strukturo področij, kar se prevede v veliko število vejitev znotraj programske kode. To lahko zelo negativno vpliva na hitrost. Če se upošteva nizko ceno odstranjevanja na nivoju kamere in veliko količino področij, ki jih ustvarita oba tipa grafov, se lahko sklepa, da pretirana razvejanost negativno vpliva na performančnost. Razlika v performačnosti med ABT in Octree drevesi je zelo majhna, čeprav ABT skoraj vedno odstrani večje število objektov. Omenjena količina pa kljub temu ni dovoljšna, da bi se razlika opazila pri hitrosti. Le pri odprtem svetu je razlika dovolj očitna, da se dejansko pozna na performančnosti.

Ogre3D velja za preizkušen in stabilen pogon, četudi ima vrsto pomanjkljivosti in hroščev, ki znajo otežiti delo razvijalcev. Podpora za DirectX 11 ni ravno stabilna in zahteva vrsto popravkov znotraj pogona. Podobno velja tudi za vtičnik

sistema portalov, ki se ni izkazal za zelo stabilnega. Največja pomanjkljivost pogona pa še vedno ostaja odsotnost urejevalnika sveta, ki bi bil prilagojen izključno za Ogre3D. Povezava je sicer možna preko orodji kot so na Maya ali 3D Studio Max, vendar je zelo okorna in zahteva veliko časa ter dodatnega dela, zaradi česar se je načeloma modelirno orodje Maya spremenilo v delujoč urejevalnik sveta.

Nalogo bi se lahko dodatno obogatilo s testiranjem vpliva dinamičnih objektov, kar bi bistveno povečalo uporabnost binarnega razdeljevanja prostora, saj je dodajanje novih objektov zelo hitro. Največja pomanjkljivost sistema portalov je ročno postavljanje konveksnih poligonov, ki je zelo zamudno in le redko optimalno. Ker je optimalno postavitev mogoče do določene mere izračunati, se uporaba portalov zelo poenostavi. Trenutna implementacija PVS grafa ne dopušča postavitev vzorčnih točk s strani uporabnika, kar pa bi lahko zelo izboljšalo konsistenco upodobljenega sveta, tudi pri manjših delitvah. Nadaljnji razvoj bi lahko omogočil kombinacije grafov scen, kjer bi lahko določen graf vseboval različne grafe scen, prilagojene za določen del sveta. Takšen način je bil na primer uporabljen v igri Doom, kjer BSP drevo vsebuje PVS graf. Najpomembnejša izboljšava je večjedrno procesiranje, ki se vedno bolj pogosto uporablja v modernih igrah in ki bi lahko izpostavilo prednost linearnega seznama pred drevesnimi strukturami, ki jih uporabljajo ostali grafi scen.

Literatura

[1] Autodesk Inc., dosegljivo na: http://www.autodesk.com/products/maya/overview, zadnji dostop: 30. 8. 2016.

[2] Area Autodesk Inc., dosegljivo na: https://area.autodesk.com/maya_anniversary/historyga=1.207787093.691506613.1460594753, zadnji dostop: 30. 8. 2016.

[3] Data-Oriented Design, dosegljivo na: http://www.dataorienteddesign.com/dodmain/node3.html, zadnji dostop: 30. 8. 2016.

[4] Daniel Collin, Culling the Battlefield, dosegljivo na: http://www.slideshare.net/DICEStudio/culling-the-battlefield-data-oriented-design-in-practice, zadnji dostop: 30. 8. 2016.

[5] Electronic Arts Inc, dosegljivo na: https://www.battlefield.com/news/battlefield-1, zadnji dostop: 30. 8. 2016.

[6] Fallout 4, Bethesda softworks LLC, dosegljivo na: https://www.fallout4.com/, zadnji dostop: 30. 8. 2016.

[7] FPS Versus Frame Time, dosegljivo na: https://www.mvps.org/directx/articles/fps_versus_frame_time.htm, zadnji dostop: 30. 8. 2016.

[8] Game developers conference 2008, The technology of uncharted: Drake's fortune, dosegljivo na: http://www.slideshare.net/naughty_dog/the-technology-of-uncharted-drakes-fortune, zadnji dostop: 30. 8. 2016.

[9] Game Developers Conference 2014, Solving visibility and streaming in the witcher 3, dosegljivo na: http://www.gdcvault.com/play/1020231/Solving-Visibility-and-Streaming-in, zadnji dostop: 30. 8. 2016.

[10] M. Laakso, Potentially Visible Set (PVS), Helsinki university of technology, dosegljivo na: http://www.tml.tkk.fi/Opinnot/Tik-111.500/2003/paperit/MikkoLaakso.pdf, zadnji dostop 30. 8. 2016.

[11] Ogre 2.0 Porting Manual (draft), dosegljivo na: http://yosoygames.com.ar/other/Ogre%202.0%20Porting%20Manual%20DRAFT.pdf, zadnji dostop: 30. 8. 2016.

[12] Ogre3D, dosegljivo na: http://www.ogre3d.org/, zadnji dostop: 30. 8. 2016.

[13] Ogremax, dosegljivo na: http://www.ogremax.com, zadnji dostop 30. 8. 2016.

[14] Overwatch, Blizzard Entertainment Inc., dosegljivo na: https://playoverwatch.com/en-gb/, zadnji dostop: 30. 8. 2016.

[15] Paul D Turner & The CEGUI Development Team, dosegljivo na: http://cegui.org.uk/, zadnji dostop: 30. 8. 2016.

[16] Rockstar Games, Grand theft auto V, dosegljivo na: http://www.rockstargames.com/V/, zadnji dostop: 30. 8. 2016.

[17] Runic Games Inc., Torchligth game, dosegljivo na: http://www.torchlightgame.com/, zadnji dostop: 30. 8. 2016.

[18] Runic Games Inc., Torchlight 2 game, dosegljivo na: http://www.torchlight2game.com/, zadnji dostop: 30. 8. 2016.

[19] Skyrim, Bethesda softworks LLC, dosegljivo na: http://www.elderscrolls.com/skyrim/, zadnji dostop: 30. 8. 2016.

[20] The witcher wild hunt 3, dosegljivo na: http://thewitcher.com/en/witcher3, zadnji dostop: 30. 8. 2016.

[21] Umbra, dosegljivo na: http://umbra3d.com/, zadnji dostop: 30. 8. 2016.

[22] Uncharted 4, Sony Computer Entertainment America LLC, dosegljivo na: `http://www.unchartedthegame.com/en-gb/`, zadnji dostop: 30. 8. 2016.

[23] Unity, dosegljivo na: `https://www.assetstore.unity3d.com/en/`, zadnji dostop: 30. 8. 2016.

[24] Unity FAQ, dosegljivo na: `http://unity3d.com/unity/faq`, zadnji dostop: 30. 8. 2016.

[25] Unity Export2 Maya, dosegljivo na: `https://www.assetstore.unity3d.com/en/#!/content/17079`, zadnji dostop: 30. 8. 2016.

[26] T. Akenine-Möller, E. Haines, N. Hoffman. Real-time rendering, 3rd edition. New York: Taylor & Francis Group, LLC, 2008, str. 647, str. 654-657, str. 658-660.

[27] M. Dickheiser. Game Programming Gems 6. Boston: Cengage Learning, 2006, str. 423-435

[28] J. Gregory. Game Engine Architecture, 2nd edition. New York: Taylor & Francis Group, LLC, 2014, str. 466-467.

[29] G. Junker. Pro Ogre3D Programming. New York: Springer-Verlag, 2006: str. 37-50, str. 77-90.

[30] E. Lengyel. Mathematics for 3D Game Programming and Computer Graphics, 3rd edition. Boston: Stacy L. Hiquet, 2011, str. 230-231.

[31] P. Peer. Tehnologija iger in navidezna resničnost, skripta, Fakulteta za računalništvo in informatika, Univerza v Ljubljani, 2010.

[32] R. Naystrom. Game Programming Patterns. Sweden: Robert Nystrom, 2014, str. 73-87.